УДК 681.321

М.Ф. КАРАВАЙ
Институт проблем управления РАН, Россия

СОВРЕМЕННЫЕ ПОДХОДЫ К ПРОБЛЕМЕ ГАРАНТОСПОСОБНОСТИ: ТЕОРИЯ И ПРАКТИКА*

Рассматриваются подходы, лежащие в основе проектирования систем обеспечения гарантоспособности и нашедшие применение в практике для критических приложений. Эти подходы находят своё воплощение на различных иерархических уровнях представления системы. Рассмотрены три таких уровня: архитектура вычислительной среды, в которую погружается прикладная задача; уровень операционной системы (ОС) или её эквивалентного монитора, реализующий системные методы отказоустойчивости; уровень аппаратуры или управляющей памяти, в значительной мере остающиеся "прозрачными" не только для пользователя, но и для ОС. Анализируется необходимость комбинации этих методов.

иерархия отказоустойчивости, парирование сбоев, отказоустойчивая память

Введение

О роли отказоустойчивости в проектировании очень точно и эмоционально высказался один из первых её "крёстных отцов": *"Отказоустойчивость является лучшей гарантией того, что системы повышенной ответственности не обманут надежд их создателей и доверия пользователей из-за неумения противостоять физическим и проектным ошибкам или ошибкам оператора, вирусам и внешним злонамеренным разрушениям информации", A. Avizienis "Toward Systematic Design of Fault-Tolerant Systems", 1997.*

Более узко отказоустойчивость можно понимать как свойство объекта правильно функционировать в течение заданного времени в условиях возникновения отказов и сбоев в работе компонент объекта (системы) и ошибок программного обеспечения. В своё время это породило новую *парадигму проектирования* надежных систем [1].

При проектировании отказоустойчивой системы все подчинено единой цели – чтобы она могла работать при наличии локальных отказов в аппаратуре или в программном обеспечении. В неизбыточной системе (и, следовательно, в неотказоустойчивой) безотказность системы $P_{нс}$ равна произведению безотказностей P_i ее компонент на интервале времени t:

$$P_{нс} = \prod_{i=1}^{N} Pi \,. \qquad (1)$$

Влияние отказоустойчивого проектирования на надежность *избыточной системы* может быть выражено следующим соотношением для показателя безотказности $P_{ис}$ [2]:

$$P_{ис} = P_{б}^{изб}\ (\text{без дефектов}) + P_{1}^{изб}\ (\text{правильной работы/дефектов}) \cdot P_{2}^{изб}\ (\text{возникновения дефекта}). \qquad (2)$$

Выражение (2) не применяется для конкретных численных расчетов, его назначение – раскрыть смысл введения отказоустойчивости и указать направление ее реализации. Составляющая $P_1^{изб}$ определяет условную вероятность правильной работы системы, несмотря на наличие в ней дефектов или ошибок. Ее взвешенное значение по вероятности P_2 возникновения этих дефектов определяет ту (потенциальную!) добавку показателя системной безотказности $P_{ис}$, которая может быть получена за счет отказоустойчивых свойств проектируемой системы.

*Работа поддержана Российским фондом фундаментальных исследований (грант 05-08-01388) и Программой № 16 ОЭММПУ РАН

Эффект гарантоспособного проектирования становится заметным или даже определяющим, если удалось спроектировать систему таким образом, что условная вероятность $P_1^{изб}$ становится близкой к 1.

Существенно, что это правило не опирается на какие-либо статистические характеристики системы или ее элементов, выражая детерминированное условие, что любой одиночный дефект, носящий не только “классический” характер, но и более сложный, например, “враждебный”, должен быть парирован. В системе, спроектированной в соответствии с этим правилом, системный отказ наступает либо при исчерпании резерва и возникновении очередного одиночного дефекта, либо при возникновении кратного (2-х и более) дефекта, либо при появлении одиночного дефекта на фоне накопившихся скрытых (латентных) дефектов.

Характер задач, решаемых системой, определяет, в каком смысле понимается отказоустойчивость. Например, для распределенной системы, в основе которой лежит обмен между абонентами, отказоустойчивость часто рассматривается как связность абонентов между собой (опуская вопросы трафика). В мультипроцессорной системе с “тесным” взаимодействием отказоустойчивость требует сохранения логической структуры решаемой задачи даже при отказах локальных компонент, что представляет более трудную задачу, чем предыдущая.

Можно говорить о разной "степени" отказоустойчивости в зависимости от того, насколько жестко определено понятие реального времени - от недопустимости даже весьма кратковременных перерывов в работе системы до заметных длительностей таких перерывов, что определяется характеристиками инерционности управляемых объектов [3].

Уровень архитектуры системы

На уровне многопроцессорной архитектуры вычислительной (управляющей) среды исследования базируются на создании такой отказоустойчивой структуры, которая позволяет перезагружать задачу при отказе произвольного активного процессора без перекомпиляции всей задачи в целом. Предполагается, что перезагрузка (если все коды не загружены во все процессоры) должна быть такой же сложности, как и начальная загрузка, не сложнее. Для достижения этого граф исходной задачи (“целевой” граф) должен вкладываться в архитектуру реализующей машины. При обнаружении отказа избыточный ресурс “целевой” реализующей системы отображается на отказавший ресурс, при этом остальные ресурсы могут изменить свои виртуальные имена, согласно которым происходит реконфигурация (перезагрузка) системы. Для этого последняя должна обладать достаточно высокой математической симметрией, т.е. порядком группы её автоморфизмов. Архитектуры систем, удовлетворяющих этим свойствам, исследовались в [4, 5]. К ним относятся шинные архитектуры, полносвязные и архитектуры со специальной симметрией.

Ещё один подход к построению отказоустойчивых архитектур многопроцессорных (многоабонентных) систем начал развиваться совсем недавно [6, 7] Это представление многоабонентной системы в виде симметричного однородного двудольного графа с вершинами двух типов X и Y. При этом вершины типа X представляют абоненты сети, а вершины типа Y – совокупность блоков локальных коммутаторов, со свойством, что каждая вершина из X связана с каждой из остальных вершин из X единственным путём длины 2, проходящим через некоторую вершину из Y. Оказывается, задача построения подобных архитектур эквивалентна задаче построения неполных уравновешенных блок-схем, хорошо изученных в дискретной математике [8]. Объектом исследований является размещение v элементов по b блокам, таким образом, что каждый блок содержит по k различных элементов, каждый элемент принадлежит r различным блокам, а каждая пара элементов появляется в λ блоках. При $v = b$ и $k = r$, $\lambda = 1$ сеть представляет собой полный коммутатор с диаметром 2, но учитывая, что передача происходит через локальные коммутаторы, а не через абонентные вершины сети, по-

следняя выглядит как квазиполнодоступная сеть, т.е. “почти” как полный коммутатор. Имея почти идентичные полному графу коммутационные возможности, *n*-вершинный квазиполнодоступный граф имеет приблизительно n√n связей по сравнению с примерно n^2 для полного графа. Поскольку полный граф – это наиболее удобная структура для реализации отказоустойчивости, и в то же время наиболее производительная, то эти свойства присущи и найденной квазиполнодоступной сети [7].

Одноканальные структуры

Существенно отличающаяся ситуация наблюдается в однопроцессорных системах. Конечно, как уже отмечалось, одноканальная архитектура не может в полной мере удовлетворить основному правилу обязательного парирования одного ***устойчивого*** отказа или программной ошибки. В ней величина вероятности безотказной работы неизбыточной системы на заданном интервале времени t $P_{нс}(t)$ должна быть близка к 1, что предопределяет применение в подобных случаях высоконадёжных компонент и гарантированных методов проектирования.

Здесь нет избыточного аппаратурного ресурса, и восстановление может происходить только по отношению к сбоям, если система имеет временную избыточность, что почти всегда справедливо. Однопроцессорные системы нередки даже в сфере ответственных применений, если сроки их активной работы ограничены. Однако, к ним предъявляются жёсткие требования по парированию сбоев в системе. Основной источник сбоев в компьютерных системах - это влияние накопленной ионизации и эффекты высокоэнергетических излучений [9].

Обычно пользуются двумя характеристиками проникающих излучений при описании радиационной стойкости электро-радиокомпонент:

- максимально *допустимая накопленная доза* (например, 150 Крад или 300 Крад для различных схем «кремний-на-сапфире», или 500 ÷ 1500 ÷ 5000 рад для обычной технологии) за время работы на орбите;
- энергетический порог образования «мягких» (SEU) и устойчивых (SEL) отказов; например, для микропроцессоров по (средним) данным фирмы Space Electronics:

MicroPr: SEU-порог 3 ÷ 9 Mev/mg/cm^2; SEL-порог свыше 60 Mev/mg/cm^2.

По другим данным (МГУ, Институт ядерной физики, 2003г.):

MicroPr (TMS320C40): SEU-порог 3 Mev/mg/cm^2; SEL- порог свыше 10 Mev/mg/cm^2;

Память SRAM: SEU-порог 1,5 ÷ 3 Mev/mg/cm^2; SEL- порог 10÷50 Mev/mg/cm^2.

Для противодействия влиянию ионизации применяется радиационно устойчивая элементная база, использующая очень дорогую технологию кремния на сапфире (КнС). Дороговизна, по-видимому, объясняется не только технологическими трудностями, но и малостью выпускаемых партий. Эта технология позволяет выдерживать большие дозы накопленной радиации в микросхемах (вплоть до 50 – 300 Крад.) без проявления устойчивого отказа. Ионизация кремниевых полупроводников вызывается потоками электронов и тяжёлых заряженных частиц, в основном, ионами, производя, при достижении определённых порогов, нарушения в распределении зарядов в кристаллах. Они вызывают как “мягкие”, (SEU – Single Event Upset -одиночный сбой), т.е. восстановимые при последующей записи сбои, так и невосстановимые (SEL – Single Event Latch – одиночная “защелка”) отказы [10, 11].

В арсенале проектировщика высоконадёжной одноканальной системы имеются следующие *системные* возможности повышения корректности вычислительного или управляющего процессов и восстановления системы при возникновении отклонений (кодовая защита памяти будет рассмотрена в следующем разделе):

1. Обнаружение возникновения сбоя в работе за счёт применения встроенных средств оперативного контроля за ходом вычислительного (управляющего) процесса. В качестве схем встроенного контроля обычно выступают аппаратные схемы провер-

ки паритета, а также известные в технической диагностике цифровых систем разнообразные контрольные схемы (checkers) для нерегулярной логики, вплоть до дублирования подсистем с целью контроля. За этим должно следовать повторное выполнение.

2. Обнаружение ошибок передачи данных по линейным цепям с помощью встроенных методов программного тестирования с целью проверки контрольной суммы линейного массива, программной проверки некоторых простых инвариантов рабочего задания, таких как число решаемых задач, соблюдение границ изменения процессов и границ разрешённых адресов и других заранее известных простых характеристик (“я живой” и т.п.).

3. Организация контрольных точек в исполняемой программе, позволяющих восстанавливать вычисления после обнаружения сбоя из ближайшей контрольной точки (“назад” или “вперёд”). Необходимы временная и программная избыточности. Естественно, такая организация требует специальных затрат на доработку программ и поглощает часть рабочих ресурсов времени. При высокопроизводительном процессоре такие потери могут оказаться вполне приемлемой платой за отказоустойчивость системы к мягким отказам. Наиболее сложным моментом здесь является реализация “приёмочного” теста, поскольку он должен быть основан на определённых прикладных знаниях об исполняемых процессах.

4. Двойной последовательный просчёт и сравнение результатов с целью обнаружения сбоя и последующего восстановления из реперной точки. Для этого необходимы избыточная производительность системы и соблюдение трафика.

5. Двойной параллельный просчёт, использующий мульти-нитиевую (multi-thread) архитектуру современных высокопроизводительных микропроцессоров для контроля вычислений. Здесь возможно ведение двух различных алгоритмов решения одной и той же задачи, либо один точный алгоритм а другой – приближённый.

6. Контроль времени (сторожевой таймер) завершения задачи и контроль за признаками завершения задачи, которые присутствуют практически во всех ОС и системных программах систем реального времени. Восстановление – из реперной точки или перезапуском задачи.

Перечисленным практически исчерпывается арсенал системных возможностей обнаружения и парирования сбоев и восстановления вычислительного процесса в одноканальной системе. Надо понимать, что устойчивые одиночные отказы будут парироваться только в памяти (EDAC). В остальных подсистемах, в принципе, возможно восстановление только от сбоев. Поскольку в основе перечисленных возможностей лежат в том или ином виде тестовые проверки (двойной счёт) или реакции схем контроля (чётность, контрольные суммы и др.), то обработчик прерываний, включённый в системную библиотеку ОС, должен учитывать специфику каждого прерывания, принимая во внимание, текущие данные - управляющие или нет, а также возможную неточность данных. О взаимовлиянии обработчика прерываний и процессов тестового и функционального диагностирования в системах реального времени можно познакомиться в [12, 13].

Все упомянутые возможности, конечно, могут работать и в многоканальных архитектурах, но в этих случаях контроль *сравнение результатов* различных каналов обеспечивает наиболее универсальный метод диверсионного контроля корректности процесса с последующим восстановлением от сбоев или тестированием для выявления отказавшего канала. Однако, надо помнить, что рано или поздно многоканальная архитектура может деградировать до одноканальной, и все перечисленные механизмы контроля и восстановления окажутся востребованными. Вот почему эти механизмы надо закладывать уже на стадии проектирования базовой одноканальной архитектуры.

Повышение защитных функций ядра ОС

Системные методы борьбы со сбоями достаточно давно и хорошо известны. Механизм действия кон-

трольных точек, промежуточных сравнений и других системных ухищрений также хорошо известен, но, к сожалению, применяется не часто. Во всяком случае, все эти действия – это зона ответственности ОС, поскольку опасно допускать пользователя до реализации системных вызовов. Хуже обстоит дело с наличием ошибок в ОС или с "мягкими" отказами, возникающими при исполнении кодов ядра ОС. Сбои в адресации при работе драйверов могут приводить к очень тяжёлым последствиям из-за обращения к несанкционированным адресам портов и устройств.

Не вызывает сомнения постулат, что операционная система должна быть надёжной и безопасной. Первое подразумевает исполнение всех предписанных функций в надлежащее время, второе требует, чтобы система не делала "глупостей", поскольку несанкционированные действия, инспирированные операционной системой, недопустимы. В основе же и того, и другого лежит одно: программные ошибки или сбои, затрагивающие коды ядра ОС.

Программные ошибки в ядре или в системных функциях ОС значительно опаснее ошибок прикладных задач. Если ОС корректна, то ошибки прикладных программ могут вызвать лишь ограниченное разрушение, что связано в системах реального времени со сравнительно небольшими размерами управляющих программ, с заранее предсказуемым характером поведения управляемой подсистемы и, главное, с изоляцией разных задач от непосредственного общения друг с другом. В ОС ситуация другая. Во-первых, со временем они очень быстро "разбухают", как тот же Linux, начавшийся от единиц тысяч строк кода, и ядро которого сейчас насчитывает более 2,5 миллионов строк. Известный специалист в области операционных систем Andrew Tanenbaum оценивает (в 2006 г.) число программных ошибок на 1000 строк кода в ядре Linux от 6 до 16 [14]. По самым консервативным оценкам ядро Linux имеет, по всей вероятности, около 15000 ошибок. Не слабо!

Исследования, доложенные на Международных симпозиумах по надёжности программного обеспечения, в последние годы показывают, что основная масса ошибок ОС, более 70%, сосредоточена в драйверах. Интенсивность ошибок здесь в 3 ÷ 7 раз выше, чем в обычных кодах. Нахождение этих ошибок сложно, поскольку они связаны с драйверами устройств, а некоторые попытки их исправления приводили к порождению ещё большего числа ошибок. Выход из подобной ситуации ищется, и в настоящее время предложено несколько подходов к повышению надёжности ОС РВ.

Первый из подходов предложен в 2005 г. группой исследователей M,Swift, B, Bershad и H. Levy. Это наиболее консервативный и "щадящий" подход, не затрагивающий монолитного ядра ОС РВ, назван Nooks (т.е. "укромный уголок" для Linux). Nooks поддерживает монолитную структуру ядра, содержащую тысячи процедур, связанных в едином адресном пространстве ядра, но фокусирует своё внимание на том, чтобы работа с драйверами устройств, а в этом основная суть проблемы, таила меньшую опасность. Суть предложения - окружить каждый драйвер защитной оболочкой, или доменом. Эта оболочка отслеживает все взаимодействия драйвера и ядра. У Nooks три основных цели:

– Защита ядра от ошибок в драйвере.

– Автоматическое восстановление при отказе драйвера.

– Минимальные изменения в существующих драйверах и в ядре.

При всякой попытке драйвера модифицировать объект ядра его оболочка копирует этот объект на собственные страницы, доступные для чтения/записи. Драйвер, при необходимости, модифицирует эту копию. Если этот процесс заканчивается успешно, то оболочка копирует модифицированный объект в ядро. При таком подходе отказ драйвера при запросе не затрагивает структуру ядра. Поскольку работа с каждым импортируемым объектом имеет свою специфику, команде разработчиков

Nooks пришлось вручную написать программы для 43 классов объектов, используемых Linux.

В Nooks добавлена концепция теневых драйверов, позволяющая приложению продолжить работу и после упомянутых отказов драйвера. Во время нормальной работы теневой драйвер ведёт журнал взаимодействий между драйвером и ядром с целью сохранения необходимых данных для возможного восстановления. В случае рестарта отказавшего драйвера теневой драйвер снабжает его необходимыми данными из лога. В этом случае ядро ОС не занимается возвратом драйвера в правильное исходное состояние для продолжения работы, это становится функцией теневого драйвера.

Другие, более радикальные подходы, в которых изменениям подвергается уже сама структура ядра ОС, выходят за рамки наших обсуждений. Этот круг вопросов повышения надёжности ОС РВ может представить интерес для ближайшего будущего, поскольку пока находится в практической доводке и может быть доступен к моменту разработки многоканальных систем РВ на базе нового отечественного кристалла 1892ВМ3Т (Multicore MC-12).

Уровень аппаратуры

Многочисленные исследования показывают, что при энергии излучений меньше 100 МэВ/мг/см^2, что типично для естественных излучений, кроме опасных вспышек на Солнце, наиболее уязвимым местом является оперативная память машины, причём не только динамическая, но и статическая. На неё приходится основная доля сбоев. По различным оценкам (часто расходящимся почти на два порядка) в космосе на высоких орбитах интенсивность SEU отказов **в не Кремний_на_Сапфире чипах** и без какой-либо пассивной защиты – от $4\cdot10^{-8}$/ бит/день до $3{,}2\cdot10^{-5}$/ бит/день, что соответственно означает для памяти в 1Мбайт **от 2 до 2 тысяч SEU отказов в неделю** [15]. Очевидно, что работать без пассивной защиты и без контроллера EDAC (обнаружение и коррекция одиночного сбоя) в этих условиях практически невозможно. Столь высокие значения “мягких” сбоев приводят как к увеличению вероятности появления двойных (и более) сбоев в одном слове памяти, не парируемых EDAC контроллером, так и к появлению катастрофических отказов (SEL), трудно поддающихся парированию. К сожалению, КнС технология, обеспечивая памяти “иммунитет” к ионизации от свободных электронов и низкоэнергетичных ионов до порога порядка 150 Крад и выше, как и дюралевый экран толщиной 8 – 12 мм., от воздействия высокоэнергетических ионов в должной мере не спасают. Поток сбоев (в памяти) даже с такой защитой остаётся неприемлемым.

Поэтому основные надежды на оперативное восстановление от **одиночных** “мягких” сбоев связываются с избыточными кодами Хэмминга и с более сложными циклическими кодами. Отсюда следует вывод, и он подтверждается на практике, что наличие контроллера кода Хэмминга в памяти системы управления – обязательно. И тем не менее, внутренне присущая памяти “неиммуность” к радиации даёт поток возможных двойных сбоев и комбинированных отказов типа двойной устойчивый отказ в слове, или сбой + устойчивый отказ в одном слове, на один-два порядка более высокий, чем одиночные отказы в остальных модулях системы.

Но всё, что было сказано выше о памяти, выдвигает к решению первую важнейшую задачу: поскольку на традиционной архитектуре памяти возникают серьёзные проблемы её отказоустойчивости, то попытаться найти новые архитектурные решения организации памяти на локальном уровне, ведущие к существенному повышению её отказоустойчивости. Упор на локальный уровень сделан по причине исчерпания возможностей системного уровня (т.е. ОС РВ), где изменения или не желательны, или даже невозможны.

Во всех известных системах появление двойного **отказа** является катастрофической ошибкой, и выход из неё зависит уже от действий на системном уровне, т.е. от операционной системы. Для повышения надёжности наименее устойчивой подсистемы управляющей системы – памяти, нужны разработки,

парирующие двойные устойчивые отказы в словах памяти. Подобные технические решения уже предложены [16], но они решают только часть задачи: восстановление от кратных сбоев всё равно возлагается на системные методы. Весьма эффективный метод для предотвращения возникновения кратных сбоев реализуется в виде программы "сборщика мусора", периодически просматривающей всю память с помощью EDAC контроллера и восстанавливающей содержимое при наличии одиночных ошибок, тем самым резко снижая вероятность появления кратных сбоев.

Выводы

Рассмотрены теоретические и практические аспекты задачи проектирования систем обеспечения гарантоспособности на трёх иерархических уровнях управляющих систем реального времени. На высшем уровне это отказоустойчивые архитектуры, в которые должны погружаться прикладные задачи. На уровне системных задач ОС это процедуры контроля и восстановления от сбоев и защиты от ошибок в действиях ОС в одноканальных системах. На аппаратурном уровне это вопросы защиты памяти от одиночных и кратных сбоев и устойчивых отказов.

Литература

1. Avizienis A., Laprie J.-C. Dependable Computing: from Concepts to Application // IEEE Trans. on Comp. – 1986. – № 5. – P. 629-638.

2. Suri N., Valter C.J., Hugu M.M. Advances in Ultra Dependable Distributed Systems // Computer Society Press. – Ca. – 1995. – P. 56-61.

3. Харченко В.С. Гарантноспособность и гарантоспособные системы: Элементы методологии // Радиоэлектроника и компьютерные системы. – 2006. – № 5. – С. 7-19.

4. Каравай М.Ф. Минимизированное вложение произвольных гамильтоновых графов в отказоустойчивый граф и реконфигурации при отказах. Ч. I. Одно-отказоустойчивые структуры // АиТ. – 2004. – № 12. – С. 159-177.

5. Каравай М.Ф. Минимизированное вложение произвольных гамильтоновых графов в отказоустойчивый граф и реконфигурации при отказах. Ч. II. Решетки и *k*-отказоустойчивость // АиТ. – 2005. – № 2. – С. 175-189.

6. Подлазов В.С., Соколов В.В. Метод однородного расширения системных сетей многопроцессорных вычислительных систем // Проблемы управления. – 2007. – № 1. – С. 37-41.

7. Каравай М.Ф, Пархоменко П.П., Подлазов В.С. Комбинаторный метод построения двудольных однородных неизбыточных квазиполносвязных графов (симметричных блок-схем) // Автоматика и Телемеханика. – 2007. – № 1. – С. 35-41.

8. Холл М. Комбинаторика. – М.: Мир, 1970. – 340 с.

9. Radiation Data // IEEE Aerospace&Electronic Systems Mag. – Apr. 2001. – N 4; Oct. 2001, N 10.

10. Radiation-Hardened FPGAs // IEEE Aerospace-&Electronic Systems Magazine. – July. 2002. – N 7.

11. Honeywell [Электронный ресурс]. – Режим доступа: www. honeywell.ru.

12. Дрозд А.В., Лобачёв М.В. Использование рабочего диагностирования при решении вычислительных задач. // Радиоэлектроника и компьютерные системы. – 2006. – № 5. – С. 142-147.

13. Saggese G.P., Wang N.J., Kalbarczyk Z.T. et al. An Experimental Study of Soft Errors in Microprocessors // IEEE. Micro . – 2005. – V.25. № 6. –P. 30-39.

14. Tanenbaum A.S., Herder J.R., Bos H. Can We Make Operating Systems Reliable and Secure? // IEEE Computer. – 2006. – № 39(5). – P. 44-51.

15. SpaceElectronics [Электронный ресурс]. – Режим доступа: www.spacelectronics.com.

16. Karavay M.F., Sinelnikov V. V. Background cache for improving memory fault tolerance // Proceedings of IEEE EAST-WEST Design&Test International Workshop. – Sochi: Sept. 15-19, 2006. – P. 24-28.

Поступила в редакцию 12.01.2007

Рецензент: д-р техн. наук, проф. В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков.