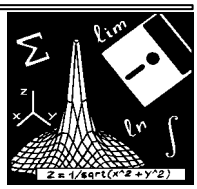
ИНФОРМАТИКА И ПРИКЛАДНАЯ МАТЕМАТИКА



УДК 004.415.53

С.С.Андреев, Ю.С.Потахин

МОДЕЛЬ ПРОЦЕССА ОТЛАДКИ ПРОГРАММНОЙ СИСТЕМЫ С ИЗМЕНЯЮЩИМСЯ ВЕКТОРОМ СОСТОЯНИЙ

Институт электронных и информационных систем НовГУ, Sergey.Andreev@novsu.ru

The issue posed in the article consists in debugging complex systems with a variable state vector. Other issues under consideration are: the function of external influence on bug reproduction time; the classic debugging model and the debugging model, which relies on saving state vector of the system.

Ключевые слова: метод сохранения вектора состояний системы, системы с изменяющимся вектором состояний

При разработке систем, работающих с большими объемами данных нестандартных форматов, важную роль играет возможность восстановления состояния системы после длительной работы (например, после нескольких дней или месяцев). В качестве примера можно привести систему видеонаблюдения, которая записывает информацию на жесткие диски компьютера в специализированную файловую систему. После длительного периода работы подобной системы есть вероятность появления дефектов, которые не проявляются на коротких сценариях. Такие дефекты могут приводить к некорректному поведению системы, а в худшем случае — даже и к некорректному завершению ее работы. Для возобновления работ над системой (как для продолжения тестирования командой тестеров, так и для отладки командой разработчиков) необходимо восстановить состояние системы на определенный момент времени, что в свою очередь требует значительных временных затрат, вплоть до приведения системы к состоянию на момент краха «с нуля». Значительные затраты времени негативно сказываются на процессе разработки системы.

Одним из методов решения данной проблемы является метод сохранения текущего вектора состояний (или контекста) тестируемой программной системы. Однако сохранение контекста только на первый взгляд кажется простой задачей. На самом же деле необходимо учитывать множество факторов, начиная со времени создания резервной копии и заканчивая типами внешних носителей информации.

Виды ошибок

В программной системе ошибка может возникать из-за разных причин — от некорректной инициализации переменной до некорректного управления памятью. Но несмотря на это виды ошибок условно можно разделить на два больших класса. Обозначим временной интервал, через который проявляется ошибка, T_{error}. Разделим все программные ошибки по параметру T_{error} на постоянные $(T_{error} = \text{const})$ и временные ($(T_{error} \neq \text{const})$.

Постоянной ошибкой считается ошибка, которая возникает в определенном компоненте через постоянный период времени T_{error} после определенных внешних воздействий на систему независимо от состояния других компонентов тестируемой системы.

Плавающая ошибка — это ошибка, проявляющаяся при одновременном возникновении двух и более простых ошибок в одном или разных взаимодействующих компонентах тестируемой системы. При этом довольно сложно понять, что именно вызвало ошибку и при каких условиях она проявляется. Обычно плавающая ошибка определяется случайными величинами — такими, как разное время задержек при передаче данных через сеть, использование псевдослучайных чисел в вычислениях и т.п., но чаще всего сложности возникают из-за невозможности повторить набор внешних воздействий (в том числе действий пользователя) в той же последовательности с теми же временными интервалами.

Функция влияния внешних воздействий на время воспроизведения ошибки

Под внешними воздействиями на программную систему *S* понимаются любые события, возникшие вне системы *S*. Примерами внешних воздействий являются:

— работа пользователя с графическим интерфейсом системы *S*;

— сигналы от внешних устройств (датчик открытия двери, датчик изменения освещенности и т.д.);

— сигнал об обнаружении движения (motion detection) на видео (как программное, так и аппаратное обнаружение);

— сигналы от других приложений.

Для учета влияния внешних воздействий введем функцию $F_{EA}\left(\vec{A}\right)$ (от англ. External Actions — внешние воздействия), которая будет отражать степень влияния внешних воздействий на время воспроизведения перехода из состояния St_A в состояние St_B. Например, если при первом запуске на переход между этими состоя-

ниями было затрачено время T_{AB}, то на воспроизведение перехода между этими состояниями может потребоваться время $T_{AB} \cdot F_{EA}\left(\overrightarrow{AB}\right)$. При этом вектор $\overrightarrow{AB}$ содержит набор воздействий, проведенных в течение времени T_{AB}, каждое из которых характеризуется временем возникновением события (или воздействия), длительностью события и сложностью его воспроизведения.

Учитывая, что количество воздействий и их длительность зависят от времени работы с системой, будем считать, что $F_{EA}\left(\overrightarrow{AB}\right) \approx F_{EA}(T_{AB})$. В случае, если на систему не влияют события извне, $F_{EA}(T) = F_{EA}(0) = 1$, т. е. время перехода между двумя состояниями будет постоянным. В общем случае на время перехода между двумя состояниями могут влиять случайные величины (ошибки пользователя при воспроизведении, разное время срабатывания датчиков и т.п.), и функция влияния внешних воздействий будет выглядеть следующим образом:

$$F_{EA}(T) = 1 + f(T),$$

где $f(x)$ — случайная величина, зависящая от T. С учетом того, что при возрастании времени возрастает вероятность ошибочного воспроизведения сценария, $f_T(x)$ можно представить как экспоненциальное распределение:

$$f(x) = \begin{cases} 0, & x \le 0, \\ k \cdot \left(1 - \lambda e^{-\lambda x}\right), & x > 0, \end{cases}$$

где λ — коэффициент невнимательности пользователя, воспроизводящего дефект (среднее количество ошибок при воспроизведении сценария в единицу времени); k — коэффициент сложности сценария (среднее количество операций в сценарии в единицу времени). Оба эти коэффициента подбираются эмпирически, в зависимости от конкретного тестового сценария и ошибки, возникающей в результате его выполнения.

Классическая модель отладки

Обычно отладка осуществляется следующим образом. При обнаружении ошибочного поведения системы программист запускает программу в режиме отладки или включает вывод отладочной информации для локализации обнаруженной проблемы. При этом программист затрачивает время T_{error} для повторного воспроизведения ошибки. Такую модель отладки будем считать *классической*. Данная модель применима для любых систем на платформе Windows.

Для систем с графическим интерфейсом необходимо повторять действия пользователя в той же последовательности, что и при возникновении ошибки. В случае, если не производилось каких-либо модификаций программы, то состояние, в котором возникла ошибка, будет воспроизведено через время $T_{error} \cdot F_{EA}(T_{error})$, при этом $F_{EA}(T_{error}) \ge 1$. Значение 1 функция принимает лишь в случаях воспроизведения ошибки с первого раза.

В общем виде время на воспроизведение плавающей ошибки $T_{reproduce}$ в классической модели отладки будет определяться формулой

$$T_{reproduce} = T_{error} \cdot F_{EA}(T_{error}).$$

На финальных стадиях проекта время воспроизведения ошибки T_{error} может быть весьма значительным. Например, в разрабатываемой компанией АстроСофт системе видеонаблюдения одна из критических ошибок проявляется лишь после месяца непрерывной работы системы, что приводит к значительным сложностям при ее воспроизведении.

Модель отладки, использующая метод сохранения вектора состояний системы

Предлагаемый метод сохранения векторов состояний («срезов») тестируемой программной системы, основанный на сохранении полного состояния (контекста) системы (т.е. состояния оперативной памяти, диска с данными и др.) на определенный момент времени, позволяет значительно сократить время воспроизведения ошибок. Принцип его достаточно прост: после старта система работает некоторое время, затем ее состояние сохраняется на специальных носителях, после чего система продолжает работать дальше.

Этот метод имеет две основные характеристики: T_{saving} — время сохранения единичного состояния системы и $T_{executing}$ — интервал времени между сохранениями состояний, в течение которого система полноценно функционирует. Время T_{saving} зависит от типа используемых носителей данных, $T_{saving} >> 0$. Например, чтобы сделать копию данных с винчестера объемом 500 Гбайт, имеющего интерфейс SATA, скорость копирования с которого составляет более 1,5 Гбит/с, потребуется 2670 секунд (около 45 минут). На момент сохранения состояния работа системы, естественно, приостанавливается.

При использовании данной методики ошибка возникает через то же время T_{error} относительно времени работы системы, но с учетом времени T_{saving} время относительно старта тестируемой системы составит величину T'_{error}:

$$T'_{error} = T_{error} + N \cdot T_{saving},$$

где $N = \left[\frac{T_{error}}{T_{executing}}\right]$ — количество сохранений состояния системы до возникновения ошибки.

Следует отметить, что метод сохранения состояний при отладке сложных систем эффективен лишь в случаях, когда накладные расходы значительно меньше времени воспроизведения ошибки: $T_{error} >> N \cdot T_{saving}$.

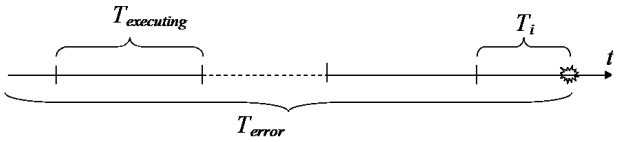
В случае обнаружения ошибки для повторного ее воспроизведения потребуется лишь восстановить последнее сохраненное состояние тестируемой системы. Время восстановления состояния системы $T_{restoring}$ обычно соизмеримо со временем сохранения копии этого состояния: $T_{restoring} \approx T_{saving}$. В этом слу-

чае для восстановления состояния на момент ошибки нам потребуется время

$$T = T_{restoring} + T_i \cdot F_{EA}(T_i),$$

где T_i — это временной интервал между последним сохранением состояния системы и моментом возникновения ошибки: $T_i = T_{error} - N \cdot T_{executing}$.

Схема взаимосвязи между интервалами $T_{executing}$, T_{error} и T_i представлена на рис.



Соотношение времени ошибки и времени ее воспроизведения

Отметим, что при сохранении состояния на определенный момент времени сохраняются также все изменения, вызванные внешними воздействиями до этого момента. Таким образом, вместо того, чтобы воспроизводить все действия пользователя за время T_{error}, достаточно лишь воспроизвести действия, выполненные за последний интервал $T_{executing}$, т.е. функция $F_{EA}(T)$ будет принимать меньшие значения:

$$F_{EA}(T_1) \le F_{EA}(T_2) \text{ при } T_1 < T_2,$$

что также сократит время воспроизведения сложных сценариев.

Сравнение моделей

Время, требуемое для обнаружения и воспроизведения дефекта, $T = T_{error} + T_{reproduce}$. Для классической модели

$$T_{KM} = T_{error} + T_{error} \cdot F_{EA}(T_{error}) = T_{error} \cdot (1 + F_{EA}(T_{error})).$$

При использовании метода сохранения состояний цикл обнаружения ошибки и ее воспроизведения составит

$$T_{MCC} = \left(T_{error} + N \cdot T_{saving}\right) + \left(T_{restoring} + T_i \cdot F_{EA}(T_i)\right).$$

Учитывая, что $T_{restoring} \approx T_{saving}$,

$$T_{MCC} = T_{error} + (N+1) \cdot T_{saving} + T_i \cdot F_{EA}(T_i).$$

В идеальном случае, если дефект воспроизведется с первого раза (т.е. $F_{EA}(T) = 1$), получаем следующие соотношения:

$$T_{KM} = T_{error} \cdot (1 + F_{EA}(T_{error})) = 2 \cdot T_{error},$$

$$T_{MCC} = T_{error} + (N+1) \cdot T_{saving} + T_i \cdot F_{EA}(T_i) = T_{error} + (N+1) \cdot T_{saving} + T_i.$$

Введем коэффициент μ, отражающий отношение времен обнаружения и воспроизведения дефекта при использовании обеих моделей:

$$\mu = \frac{T_{KM}}{T_{MCC}} = \frac{2 \cdot T_{error}}{T_{error} + (N+1) \cdot T_{saving} + T_i}.$$

В качестве примера произведем сравнение для тестируемой системе *S*, состояния которой сохраняются через каждые 3 часа, время сохранения состояния — 25 минут, а через 200 часов (8 дней) происходит ошибка. Для системы *S* значение μ, вычисленное с указанными выше допущениями, составит 1,74. В таблице приведены значения μ для нескольких воспроизведений указанной ошибки в системе *S*.

Сравнение рассматриваемых подходов для нескольких воспроизведений ошибки в системе *S*

Количество восстановлений состояния системы	1	2	3	4	5
μ, не менее	1,74	2,58	3,41	4,22	5,01

Заключение

Таким образом, при отладке сложной программной системы с использованием метода сохранения вектора состояний время на обнаружение и воспроизведение дефекта сокращается минимум в разы. Разработка программного комплекса с применением данной модели позволит эффективнее использовать время и оборудование для стабилизации работы программных систем с изменяющимся вектором состояний, что поможет сохранить софтверным компаниям финансовые средства и уменьшить время разработки подобных систем.