XXXII Неделя науки СПбГПУ. Материалы межвузовской научно-технической конференции. Ч.V : С.20-21

УДК 681.3.06

К.Г.Зыбин (асп., каф. ИУС), В.П.Котляров, к.т.н., проф.

АВТОМАТИЗАЦИЯ НАХОЖДЕНИЯ ОШИБОК РАБОТЫ С ПАМЯТЬЮ В ПРОГРАММАХ, НАПИСАНЫХ НА С И С++

Языки программирования С и С++ – одни из наиболее широко используемых языков в современной индустрии программного обеспечения. Данные языки имеют развитую поддержу для многих платформ и множество инструментальных средств отладки. При написании программ на С активно используется статическая и динамическая память. При этом ответственность за использование и освобождение памяти ложится на программиста. Такой подход позволяет создавать гибкие и мощные программные продукты, однако вероятность возникновения ошибок при таком использовании памяти очень велика. Ошибки, связанные с обработкой памяти, как правило, проявляются в виде утечек памяти и доступа к памяти по неправильному адресу. Ошибки последнего типа могут привести к сбою в работе всей системы и ошибкам на шине данных. Поиск ошибок работы с памятью занимает значительную часть времени и существенно удлиняет процесс отладки программы. Поиск утечек памяти вручную, как показывает практика, – вообще малоэффективная операция. Для предотвращения утечек памяти возможно написание собственных процедур выделения памяти [1–3], однако данный подход не предотвращает ошибок с неправильными указателями и ссылками.

Таким образом, для поиска ошибок необходим дополнительный монитор памяти, который бы следил, за ее распределением. В одном из подходов [4] для всей памяти, выделенной программе операционной системой, создается таблица состояния. В этой таблице для каждой ячейки выделено 2 бита, которые реализуют два флага: один флаг сигнализирует о том, что данная ячейка была выделена, другой – о том, что ячейка инициализирована. В двух битах хранится одно из состояний конечного автомата, который описывает поведение данной ячейки памяти. Всего существует 4 состояния автомата:

- “Красная память” – ячейка не выделена и не проинициализирована. Доступ к такой памяти запрещен.
- “Желтая память” – ячейка выделена, но не проинициализирована. К такой памяти доступ на чтение запрещен.
- “Зеленая память” – ячейка выделена и инициализирована. Доступ к такой памяти разрешен.
- ”Синяя память” – ячейка, возвращена менеджеру памяти (выполнена С/С++ команда free или delete [6]). Доступ к такой памяти запрещен.

При любом обращении к памяти монитор проверяет ее состояние в соответствии с данной таблицей, и если доступ к памяти запрещен, то он выдает ошибку и локализует место обращения к памяти. После выполнения программы монитор анализирует состояние таблицы и если найдены ячейки желтой или зеленой памяти – это свидетельствует об утечках памяти. Снабдив данный монитор модулем, который ведет журналирование мест программы, откуда производилась модификация ячеек памяти, получаем удобный отчет об ошибках и конкретных строчках в коде, где они произошли. Данный подход позволяет находить все ошибки, связанные с динамической памятью.

Для нахождения ошибок, связанных со статической памятью, таких как выход за границы массива, необходимо создавать специальные ячейки-маркеры до и после каждого

статического массива и переменной. Ячейки-маркеры – это обычные ячейки памяти, которая помечена как “красная память”. При этом если программа производит доступ за границу массива, она пытается получить доступ к “красной памяти”, и монитор сигнализирует об ошибке.

Описанный подход не требует модификации исходного кода программы, но требует вмешательства на этапе сборки исполняемого файла из объектных файлов. Подобный монитор реализован в продукте фирмы Rational – PurifyPlus.

Использование подобного монитора памяти позволяет находить ошибки на этапе выполнения программы, что предполагает наличие тестового сценария. Максимальная эффективность мониторинга памяти достигается при максимальном покрытии программы [5]. Таким образом, использование монитора необходимо вместе с использованием другого инструмента для подсчета покрытия, например, любого стандартного profiler-а. Это позволяет анализировать результаты и измерять качество продукта (предположительное число не найденных ошибок)

Предложенный подход был применен в одном из проектов разработки программного продукта (язык разработки ANSI-C), при этом проводилось сравнение с процессом отладки предыдущих версий этого же продукта. В результате было найдено 12 ошибок унаследованных с предыдущих версий, кроме того, время отладки и поиска ошибок уменьшилось примерно в 20 раз.

ЛИТЕРАТУРА:

1. Eckel, Bruce. Thinking in C++. // Upper Saddle River, NJ: Prentice Hall, 2000.
2. Murphy, Niall. "Assert Yourself," // Embedded Systems Programming, May 2001.
3. Murphy, Niall. “More on memory leaks”. // Embedded Systems Programming, March 2002.
4. Rational Purify installing and getting started // 1999, 2001 Rational Software Corporation.
5. Steve Cornett. Code Coverage Analysis. // Bullseye Testing Technology, 2002 (http://www.bullseye.com/webCoverage.html).
6. Бьерн Страуструп. Язык программирования C++. Бином 1999.
7. Теренс Чан. Системное программирование на C++ для Unix. - Киев BHV, 1999.