УДК 004.05

Г. П. Можаров, И. В. Парфилов

ПОДХОД К ПОИСКУ ВЗАИМНЫХ БЛОКИРОВОК В МНОГОПОТОЧНОМ ПРОГРАММНОМ ОБЕСПЕЧЕНИИ С ПОМОЩЬЮ ВЕРИФИКАТОРА SPIN

Рассмотрена задача поиска потенциальных взаимных блокировок в многопоточном программном обеспечении. Предложен подход к выявлению потенциальных взаимных блокировок в многопоточных программных комплексах на основе метода Model Checking. Описано, каким образом основные примитивы синхронизации могут быть представлены на входном языке верификатора SPIN. Приведен пример моделирования и выявления взаимной блокировки в многопоточной программе.

E-mail: arkandreev@gmail.com

Ключевые слова: *взаимная блокировка, синхронизация потоков, проверка модели.*

В настоящее время в связи с развитием многоядерных микропроцессорных технологий возрастает актуальность параллельного программирования. При разработке сложных многопоточных программных средств важно учитывать ряд возможных проблем, которые в дальнейшем могут привести к их нестабильной работе. Одним из признаков нестабильной работы многопоточной программы является возникновение взаимной блокировки потоков в процессе их выполнения. Взаимные блокировки происходят в результате некорректного использования средств синхронизации потоков, когда имеется группа потоков, каждый из которых пытается получить эксклюзивный доступ к некоторым ресурсам и претендует на ресурсы, принадлежащие другому потоку. В итоге все они оказываются в состоянии бесконечного ожидания доступа к ресурсу.

Основной проблемой выявления ошибок в использовании средств синхронизации стандартными методами (в результате динамического анализа [1, 2]) является сильная зависимость возникновения ситуаций взаимных блокировок от динамики выполнения программы в конкретной программно-аппаратной среде. Это свойство не позволяет создать эффективные алгоритмы выявления взаимных блокировок на этапе тестирования программного обеспечения (ПО). Практически об устранении взаимных блокировок необходимо заботиться на этапе детализированного проектирования системы. Для решения проблемы возникновения взаимных блокировок на этапе детализированного проектирования целесообразно использовать технологии верификации, основанные на методе Model Checking [3, 4]. Верификация программных систем с помощью метода Model Checking состоит из

построения формальной модели ПО и проверки заданных свойств на данной модели с помощью автоматизированных средств.

Рассмотрим подход к выявлению потенциальных взаимных блокировок в многопоточных программах, основанный на методе Model Checking. Он заключается в построении схемы использования потоками средств синхронизации, спецификации этой схемы на языке Promela и ее анализа с помощью верификатора SPIN [5].

Представление основных средств синхронизации на языке Promela. SPIN — свободно распространяемый пакет программ для формальной верификации систем [5]. Этот пакет используется главным образом для верификации моделей асинхронных программных систем и, в частности, коммуникационных протоколов. Он способен выявлять в моделях программ взаимную блокировку, недостижимые участки кода, а также определять выполнимость LTL-спецификаций. С помощью системы SPIN выполняется проверка не самих программ, а их моделей. Для построения модели по исходной программе или алгоритму пользователь (обычно вручную) строит представление этой программы на С-подобном входном языке пакета SPIN, носящем название Promela (Protocol Meta Language). Это представление (программу на языке Promela) можно считать моделью верифицируемой программы. Конструкции языка Promela просты, они имеют ясную и четкую семантику, позволяющую перевести (транслировать) любую программу на этом языке в систему переходов с конечным числом состояний, которая представляет собой модель переходов подлежащей верификации программы или алгоритма.

Анализ средств синхронизации основных операционных систем и языков программирования [6] показывает, что все предоставляемые средства синхронизации могут быть отнесены к одному из трех примитивов синхронизации, таким как исключающий семафор (мьютекс), сигнальная переменная без памяти (условная переменная) и сигнальная переменная с памятью (семафор). Основные операции над примитивами синхронизации представлены в табл. 1.

Для верификации моделей многопоточных программ с помощью программного средства SPIN, необходимо представить выделенные базовые примитивы синхронизации на языке Promela.

Сигнальную переменную с памятью можно реализовать на языке Promela следующим образом:

```
byte semA=2;
active proctype P()
        {
        /* Уменьшение счетчика сигнальной переменной */
        atomic
```

Таблица 1

Примитивы синхронизации и операции над ними

Примитив синхронизации	Операции над примитивом
Исключающий семафор (мьютекс)	Захват
	Освобождение
Сигнальная переменная без памяти	Ожидание сигнала
	Посылка сигнала
	Широковещательная посылка сигнала
Сигнальная переменная с памятью	Уменьшение счетчика
	Увеличение счетчика

```
            {
                semA>0;
                semA--
            }
            /* Уменьшение счетчика сигнальной переменной */
            atomic
            {
                semA>0;
                semA--
            }
            printf("MSC: Защищенный сигнальной переменной
блок P\n");
            /* Увеличение счетчика сигнальной переменной */
            semA++;
            semA++;
    }
```

В данном примере значение счетчика сигнальной переменной с памятью хранится в глобальной переменной semA. Операция уменьшения счетчика реализована с помощью блока

```
            atomic
            {
                semA>0;
                semA--
            }
```

Реализация данной операции потребовала использования ключевого слова atomic, позволяющего объединять команды в неделимые блоки. При верификации данный блок может быть выполнен, если выполнено условие $semA > 0$, в противном случае процесс, содержащий этот блок перейдет в состояние ожидания. Следует также отметить,

что данный блок при верификации принимается за одно состояние системы, что позволяет уменьшить число глобальных состояний модели.

Операция увеличения счетчика сигнальной переменной с памятью реализована командой semA++.

Реализация исключающих семафоров на языке Promela эквивалентна реализации семафоров для случая, когда начальное значение счетчика равно единице. В данном случае операция уменьшения счетчика сигнальной переменной с памятью будет эквивалентна операции захвата исключающего семафора, а операция увеличения счетчика сигнальной переменной с памятью будет эквивалентна операции освобождения.

Сигнальные переменные без памяти можно реализовать на языке Promela следующим образом:

```
chan condvar = [256] of { bit };
bit a =1;
byte condvarReaders = 0;
byte broadcastCounter=0;
active proctype W()
{
        printf("MSC: P(): Ожидания сигнала \n ");
        /* Операция ожидания сигнала */
        condvarReaders++;
        atomic
        {
            condvar ? a;
            condvarReaders–;
        }
        printf("MSC: P(): Сигнал получен \n");
}
active proctype E()
{
        /* Посылка сигнала */
        if  :: (condvarReaders > 0) -> condvar ! a;
            :: (condvarReaders <= 0) -> ;
        fi;
        printf("MSC: Q(): Посылка сигнала \n");
}
active proctype B()
{
        /* Широковещательная посылка сигнала */
        atomic
        {
```

```
                broadcastCounter=condvarReaders;
                do :: (broadcastCounter >0) -> condvar ! a;
                                                broadcastCounter–;
                   :: (broadcastCounter<= 0) ->break ;
                od;
            }
            printf("MSC: B(): Широковещательная посылка \n");
}
```

Основой для реализации сигнальной переменной без памяти на языке Promela служит такое средство взаимодействия процессов, как канал сообщений. Каналы сообщений используются для моделирования передачи данных от одного процесса к другому. Они объявляются локально или глобально при помощи служебного слова chan:

```
chan server = [16] of { short }
```

Здесь объявлен канал server с буфером 16, т.е. этот канал может хранить до 16 сообщений типа short, т.е. в канале может находиться не более 16 непрочитанных сообщений.

Для работы с каналами существуют операторы посылки и получения сообщений. Оператор "!" посылает сообщение со значением переменной expr в канал server, добавляя это значение к концу очереди в канале:

```
server ! expr
```

По умолчанию оператор выполняется, если канал назначения не переполнен, в противном случае он блокируется. Каналы передают сообщения в порядке FIFO: первым вошел — первым вышел.

Оператор приема сообщения "?" принимает сообщение из начала буфера канала и сохраняет его в переменной msg:

```
server ? msg
```

Выполнение приема сообщения возможно только в случае, если канал не пуст.

В представленной реализации сигнальной переменной без памяти создается три процесса. Процесс W выполняет операцию ожидания сигнала, процесс E выполняет операцию посылки сигнала для сигнальной переменной без памяти, процесс B выполняет операцию широковещательной посылки сигнала для сигнальной переменной без памяти. Операция ожидания сигнала реализована как операция чтения сообщения из канала. При отсутствии сообщений в канале процесс переходит в состояние ожидания сообщения. Операции посылки сигнала и широковещания реализованы как операции отправки сообщения в канал. В случае широковещания число сообщений, посылаемых в канал, равно числу процессов, находящихся в состоянии ожидания

сигнала от сигнальной переменной без памяти. Благодаря этому все ожидающие процессы продолжают свою работу.

SPIN поддерживает верификацию следующего набора общих свойств, называемых в SPIN базовыми:

- из класса свойств безопасности:
  - проверка сохранения локальных инвариантов, описанных с помощью оператора assert;
  - проверка на наличие некорректных конечных состояний, т.е. обнаружение взаимных блокировок процессов;
- из класса свойств живости:
  - проверка отсутствия бесконечных циклов, не содержащих операторов, помеченных меткой progress;
  - проверка отсутствия циклов с бесконечно частым выполением операторов с меткой accept.

Таким образом, обнаружение взаимных блокировок в модели многопоточной программы в SPIN происходит вследствие проверки модели на наличие некорректных конечных состояний.

Пример выявления взаимной блокировки в модели многопоточной программы. Для демонстрации возможностей по выявлению взаимных блокировок в моделях многопоточных программ рассмотрим следующий пример. Пусть логика использования средств синхронизации в программе, выполняемой в двух потоках, построена таким образом, что возможна взаимная блокировка потоков на этапе их выполнения (табл. 2). Предположим, что оба потока хотят получить эксклюзивный доступ к двум разделяемым ресурсам A и B. Для синхронизации доступа к данным ресурсам используются операции захвата исключающих семафоров, каждый из которых соответствует одному ресурсу.

Таблица 2

Очередность выполнения потоков, приводящая к взаимной блокировке

Шаг	Поток 1	Поток 2
0	Хочет получить доступ к ресурсам A и B, начинает с A	Хочет получить доступ к ресурсам A и B, начинает с B
1	Получает доступ к ресурсу A	
2		Получает доступ к ресурсу B
3		Ожидает освобождения ресурса A
4	Ожидает освобождения ресурса B	
5	Взаимная блокировка	

Для моделирования описанной многопоточной программы на языке Promela необходимо создать два процесса, каждый из которых пытается осуществить захват двух исключающих семафоров. Порядок

захвата исключающих семафоров в данных процессах различен. Далее приведен исходный код модели на языке Promela:

```
byte semA=1;
byte semB=1;
active proctype P()
{
        printf("MSC: Noncritical section P");
        atomic
        {
            semA>0;
            semA--
        }
        atomic
        {
            semB>0;
            semB–
        }
        printf("MSC: Critical section P");
        semB++;
        semA++;
}
active proctype Q()
{
        printf("MSC: Noncritical section Q ");
        atomic
        {
            semB>0;
            semB--
        }
        atomic
        {
            semA>0;
            semA--
        }
        printf("MSC: Critical section Q");
        semA++;
        semB++;
}
```

После проведения анализа составленной модели с помощью верификатора SPIN был получен отчет, содержащий информацию об ошибке в модели (рис. 1).

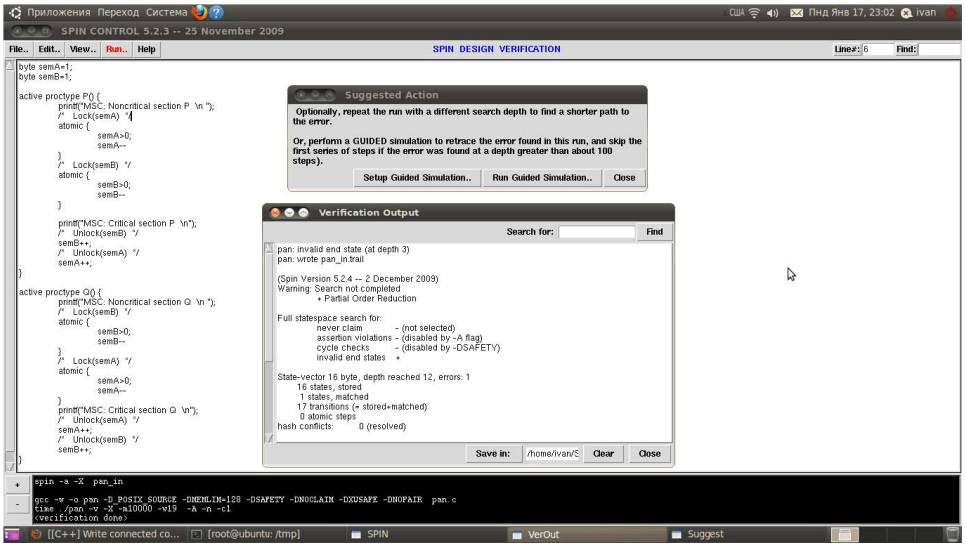

Рис. 1. Отчет о верификации модели

Одним из преимуществ подхода к верификации на основе метода Model Checking является то, что в случае невыполнения требуемых свойств будет предоставлен контрпример, нарушающий выполнение заданных свойств. Этот контрпример весьма полезен при поиске ошибок в реальной системе. При верификации свойства отсутствия взаимных блокировок в модели многопоточной программы таким контрпримером является очередность выполнения потоков, приводящая к их взаимной блокировке. На рис. 2 показана динамика работы потоков, при которой происходит взаимная блокировка.

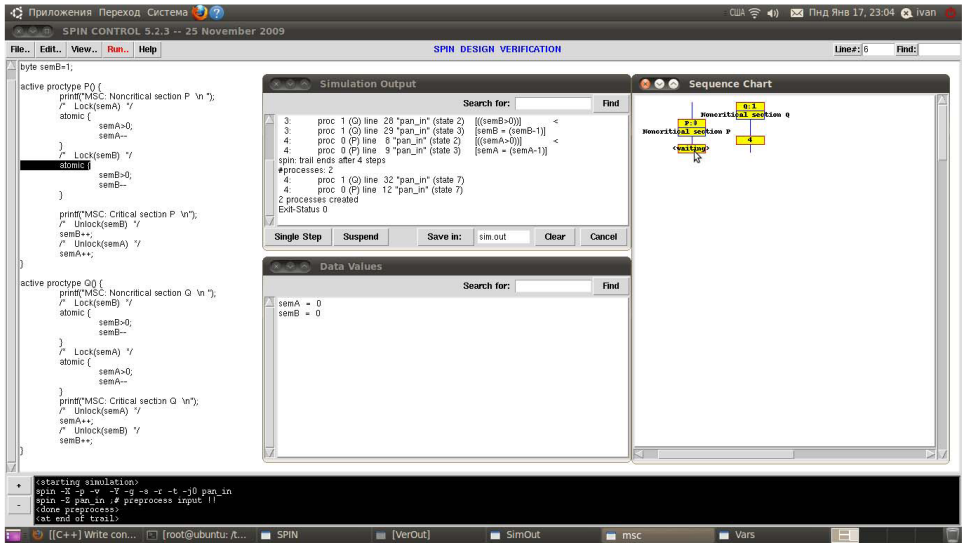

Рис. 2. Контрпример

Заключение. Предложен подход к выявлению потенциальных взаимных блокировок в многопоточных программных комплексах на основе метода Model Checking. Данный подход может применяться архитекторами и разработчиками многопоточных программных комплексов для повышения их качества благодаря представленным реализациям базовых примитивов синхронизации на входном языке верификатора SPIN и примеру использования данного подхода. К недостаткам подхода можно отнести его применимость только для сравнительно небольших программных систем. Действительно, несмотря на то, что разработано множество приемов для сокращения числа состояний в модели системы, таких как редукция частичных порядков [7], большинство из них не оказывают должного эффекта в данной постановке задачи, поскольку используют свойство независимости событий, выполняемых параллельно. Два события считаются независимыми, если при любом порядке их осуществления будет достигнуто одно и то же глобальное состояние. В данной постановке задачи события, выполняемые параллельно, не являются независимыми, поскольку именно ошибочный порядок выполнения операций доступа к средствам синхронизации и приводит к взаимной блокировке.

СПИСОК ЛИТЕРАТУРЫ

1. Bensalem S. and Havelund K. Dynamic deadlock analysis of multithreaded programs // In Shmuel Ur, Eyal Bin, and Yaron Wolfsthal, editors, Haifa Verification Conference. – 2005. – Vol. 3875. – P. 208–223.
2. Harrow J. Runtime checking of multithreaded applications with visual threads // SPIN Model Checking and Software Verification. – 2000. – Vol. 1885. – P. 331–342.
3. Кларк Э., Грамберг О., Пелед О. Верификация моделей программ: Model Checking. – М.: МНЦМО, 2002.
4. Карпов Ю. MODEL CHECKING. Верификация параллельных и распределенных программных систем. – СПб.: БХВ-Петербург, 2010.
5. Holzmann G. Spin model checker: primer and reference manual. – NJ: Addison-Wesley Professional, 2003.
6. Эхтер Ш., Робертс Д. Многоядерное программирование. – СПб.: Питер, 2010.
7. Peled D. Combining partial order reductions with on-the-fly Model Checking // J. of Formal Methods in Systems Design, 8. 1996. – P. 39–64.

Статья поступила в редакцию 15.12.2011