ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ВШ программной инженерии

ПОЛИТЕХ
Санкт-Петербургский
политехнический университет
Петра Великого

КУРСОВАЯ РАБОТА
по дисциплине «Распределенные алгоритмы»

«Моделирование и верификация распределенных алгоритмов»
Слабый приоритет писателя, SW

Студентки группы 3530202/00201 — Жук А.С.
Козлова Е.А.

Руководитель — Шошмина И.В.

Санкт-Петербург
2023 г.

Содержание

Введение

В рамках данной курсовой работы рассматривается проблема взаимно исключающего доступа к общему ресурсу в распределенных системах. Решением является разрешать доступ к общему ресурсу только тогда, когда процесс находится в определенном сегменте кода, называемым критической секцией. Условиями корректности данной проблемы являются: взаимное исключение, отсутствие блокировок и отсутствия голодания.

Постановка задачи

Algorithm 7.4: Readers and writers with a monitor

```
monitor RW
   integer readers ← 0
   integer writers ← 0
   condition OKtoRead, OKtoWrite

   operation StartRead
      if writers ≠ 0 or not empty(OKtoWrite)
         waitC(OKtoRead)
      readers ← readers + 1
      signalC(OKtoRead)

   operation EndRead
      readers ← readers − 1
      if readers = 0
         signalC(OKtoWrite)

   operation StartWrite
      if writers ≠ 0 or readers ≠ 0
         waitC(OKtoWrite)
      writers ← writers + 1

   operation EndWrite
      writers ← writers − 1
      if empty(OKtoRead)
         then signalC(OKtoWrite)
         else signalC(OKtoRead)
```

reader		writer	
p1	RW.StartRead	q1	RW StartWrite
p2	read the database	q2	write to the database
p3	RW.EndRead	q3	RW.EndWrite

Рисунок 1

Рассмотрим задачу 7.4 Читатели и писатели с монитором, взятую из книги Ben-Ari[1]. Есть несколько потоков, которые пытаются получить доступ к общему ресурсу. Проблема этой задачи аналогична проблеме взаимно исключающего доступа для n процессов, в которой несколько процессов конкурируют за доступ к критической секции.

Однако в этой задаче мы делим процессы на два класса:

1) Процессы читатели, которые исключают писателей, но не других читателей

2) Процессы писатели, которые исключает как читателей, так и других писателей

Необходимо решить эту задачу с использованием мониторов (SW) при условии того, что писатели имеют слабый приоритет.

Таким образом, задачами курсовой работы являются:

1. Построить формальную модель с заданными свойствами на PROMELA.

PROMELA (Process or Protocol Meta Language)[2] - это язык моделирования и верификации. Язык позволяет динамически создавать параллельные процессы для моделирования, например, распределенных систем.

2. Сформулировать требования к модели. В данной работе требования выражаются в виде формул линейной темпоральной логики (LTL).

3. Произвести верификацию модели.

В данной работе верификация производится с помощью Spin[2]. Spin — свободно распространяемый пакет программ для формальной верификации систем. Он разработан в исследовательском центре Bell Labs Джерардом Холзманном. Spin широко применяется как в обучении студентов методам верификации, так и в промышленности для формального анализа свойств разрабатываемых протоколов и бортовых систем.

Решение

За основу решения был взят алгоритм 7.4 Readers and writers with a monitor, взятую из книги Ben-Ari.

Процесс init запускает несколько процессов-читателей и процессов-писателей. И читатели, и писатели в бесконечном цикле проходят препротокол, критическую секцию и постпротокол.

```
proctype reader (int i){
    do
    ::
    wantR[i] = true;
    StartRead();
    critR[i] = true;
    crR++;
    crR--;
    critR[i] = false;
    EndRead();
    wantR[i] = false;
    od;
}

proctype writer (int i){
do
    ::
    wantW[i] = true;
    StartWrite();
    critW[i] = true;
    crW++;
    crW--;
    critW[i] = false;
    EndWrite();
    wantW[i] = false;
od;
}
```

Используются мониторы со следующими параметрами:

```
typedef Monitor {
    int readers;    /*количество читателей в критической секции */
    int writers;    /*количество писателей в критической секции */
    Condition OKtoRead;     /*условная переменная для блокировки читателей до тех пор пока не будет "можно читать" */
    Condition OKtoWrite;    /*условная переменная для блокировки писателей до тех пор пока не будет "можно писать" */
```

```
}

typedef Condition {
    bool gate;
    int waiting;
}
```

В функции wait_OKtoRead мы блокируем процесс, добавляя его в “очередь” командой RW.OKtoRead.waiting++, освобождаем монитор и затем ждем, когда наступит наша очередь (!waitR[i]). После этого, когда мы получим сигнал от другого процесса (RW.OKtoRead.gate), мы оказываемся в мониторе выходим из состояния ожидания.

Функция wait_OKtoWrite работает аналогично предыдущей.

```
inline wait_OKtoRead(i){
atomic {
    RW.OKtoRead.waiting++;
    lock = false;
    !waitR[i];
    RW.OKtoRead.gate;
    RW.OKtoRead.gate = false;
    RW.OKtoRead.waiting--;
}
}
inline wait_OktoWrite(i){
atomic {
    RW.OKtoWrite.waiting++;
    lock = false;
    !waitW[i];
    RW.OKtoWrite.gate;
    RW.OKtoWrite.gate = false;
    RW.OKtoWrite.waiting--;
}
}
```

В функции signalC(C) мы проверяем, есть ли ожидающие процессы. Если да, то разблокируем процесс и затем ждем возможность зайти в монитор.

```
inline signalC(C) {
atomic {
    if
    ::(C.waiting > 0) ->
        C.gate = true;
```

```
        !lock;
        lock = true;
    ::else;
    fi;
}
}
```

Также были реализованы протоколы Start Read/Write и End Read/Write. Каждая из этих функции начинается с попадания в монитор, благодаря функции enterMon(), и заканчивается выходом из монитора, функцией leaveMon().

```
inline enterMon(){
atomic{
    !lock;
    lock = true;
}
}

inline leaveMon(){
    lock = false;
}
```

Читатель при вызове StartRead() в массив waitR записывает количество ожидающих писателей пред ним и после проверяет, есть ли писатели в критической секции или ожидающие писатели. Если да, то ждет OKtoRead. После этого читатель, при необходимости, посылает сигнал другому читателю. Затем читатель уменьшает все значения массива waitW на 1 (но не меньше 0), увеличивает счетчик RW.readers и идет в критическую секцию.

Читатель при вызове EndRead() уменьшает счетчик и, если был последним, то посылает сигнал OKtoWrite.

Писатель при вызове StartWrite() в массив waitW записывает общее количество ожидающих процессов перед ним. Если кто-то есть в критической секции, то ждет OKtoWrite. Затем писатель уменьшает все значения массивов

waitW и waitR на 1 (но не меньше 0), увеличивает счетчик RW.writers и идет в критическую секцию.

Писатель при вызове EndWrite() уменьшает счетчик. Затем он проверяет, есть ли читатели, у которых в массиве waitR стоит 0 (то есть они первые в очереди за нами). Если да, то отправляет сигнал OKtoRead, иначе OKtoWrite.

Таким образом, мы реализовали слабый приоритет писателя, т.к. читатели, стоящие в очереди после писателей, войдут в критическую секцию только после них.

Верификация

1. Проверка свободы от взаимной блокировки.

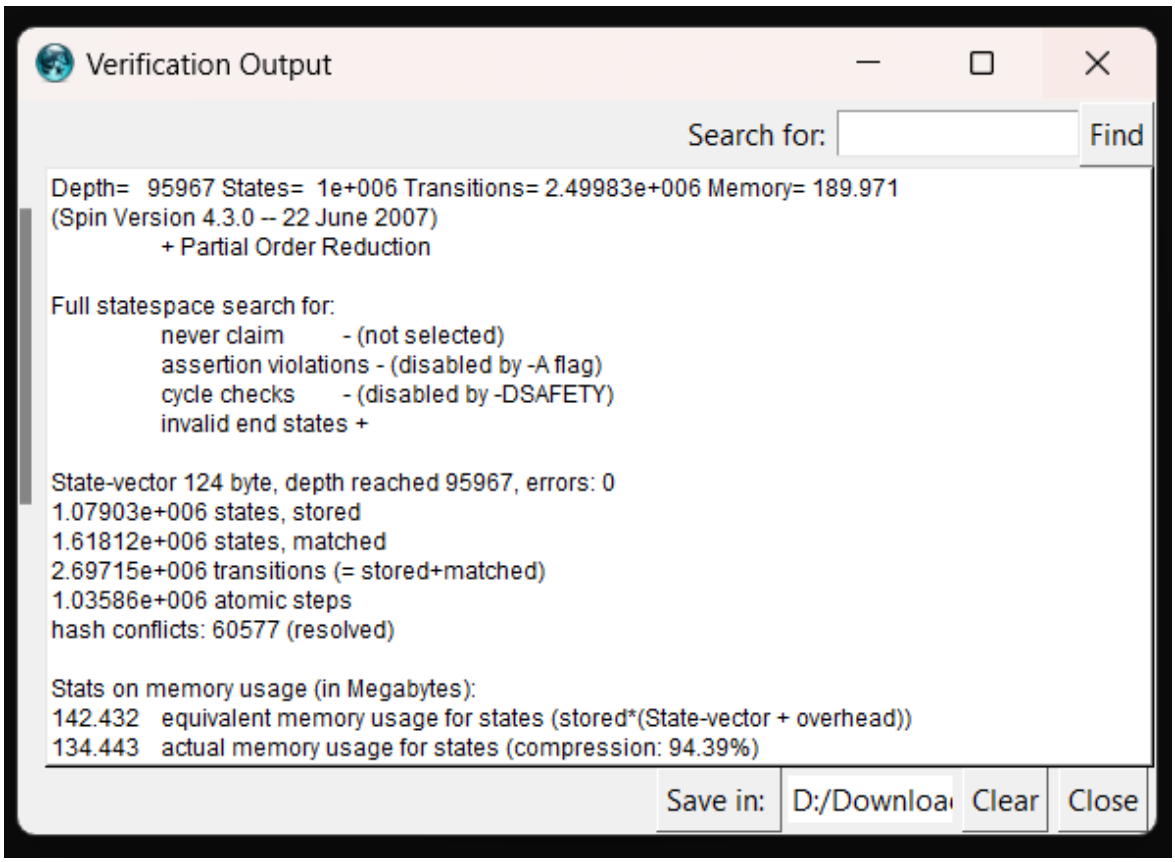

Рисунок 2.Верификация свойства отсутствия взаимной блокировки

2. Проверка на взаимоисключающий доступ.

[](RW.readers == 1 -> RW.writers == 0) && (RW.writers <= 1) && (RW.writers == 1 -> RW.readers == 0))

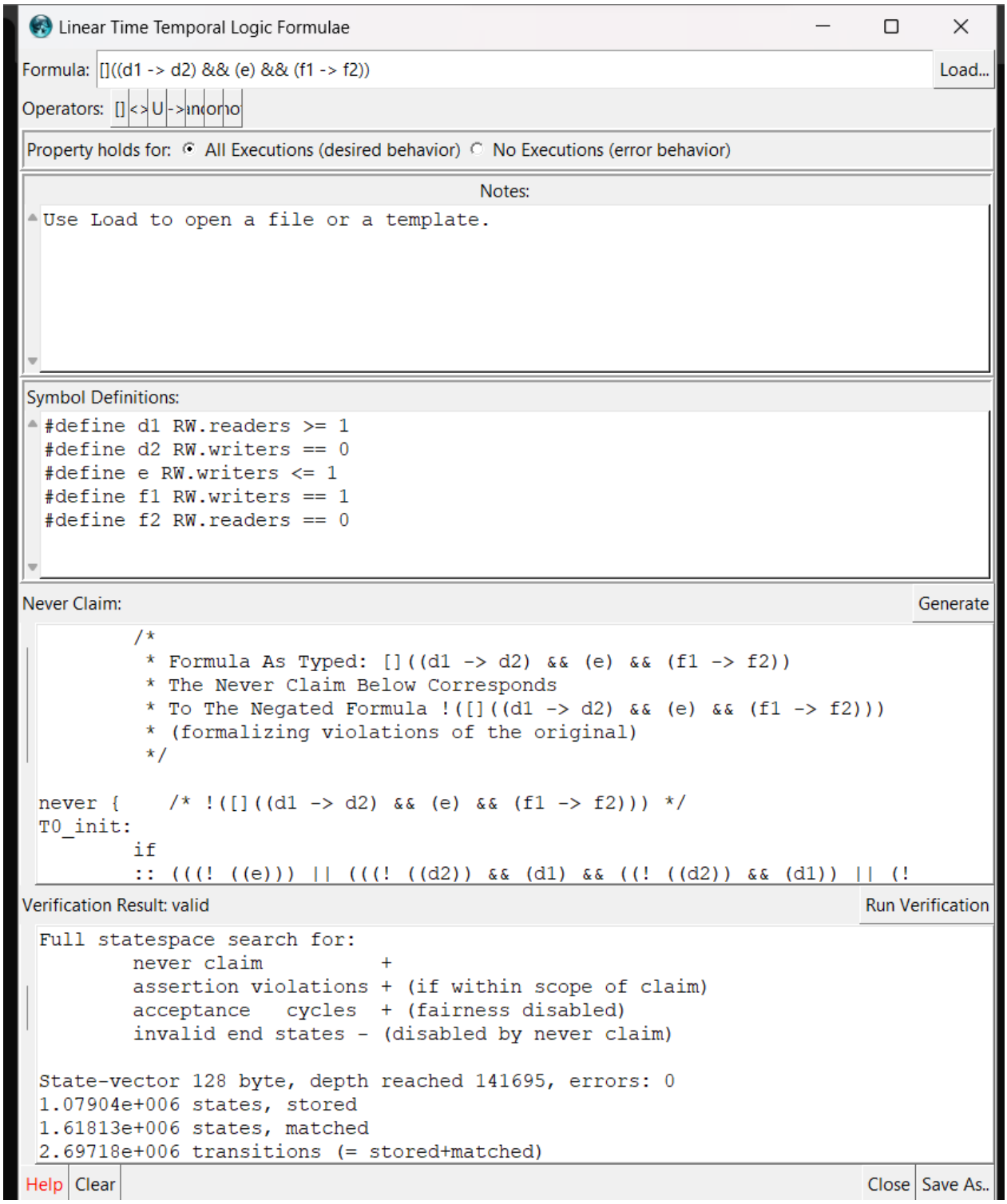

Рисунок 3. Верификация свойства взаимоисключающего доступа

3. Проверка свободы от голодания для читателей

[](wantR[0] -> <>critR[0]) && [](wantR[1] -> <>critR[1])

Свойство не выполняется, возможно голодание читателей.

4. Проверка свободы от голодания для писателей

[](wantW[0] -> <>critW[0]) && [](wantW[1] -> <>critW[1])

Свойство не выполняется, возможно голодание писателей.

Список литературы

1. M.Ben-Ari Principles of Concurrent and Distributed Programming - 2006
2. http://spinroot.com/spin/Doc/Spin_tutorial_2004.pdf Документация Promela [Дата обращения – 24.05.2023]
3. https://people.cs.rutgers.edu/~pxk/417/notes/10-mutex.html Синхронизация процессов (статья) [Дата обращения – 24.05.2023]
4. https://ru.wikipedia.org/wiki/Задача_о_читателях-писателях Википедия – свободная энциклопедия [Дата обращения – 24.05.2023]
5. https://ru.wikipedia.org/wiki/Монитор_(синхронизация) Википедия – свободная энциклопедия [Дата обращения – 24.05.2023]