МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

**Инженерно-технологическая академия**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра Математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

по лабораторной работе № 2

по курсу «Методология программной инженерии и верификация программного обеспечения»

Выполнил:

студенты группы КТмо1-3

Шепель И. О.

Куприянова А. А.

Проверил:

преподаватель каф. МОП ЭВМ

Пирская Л. В.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2016 г.

Таганрог 2016

**Цели работы:**

1. Разработать алгоритм взаимного исключения с центральным координатором и взаимодействием через разделяемую память и с помощью каналов.

2. Проверить свойства безопасности и живучести разработанных моделей, убедиться в том, что выполняется FIFO-справедливость для моделей.

**Выполнение работы:**

1. Разработка модели алгоритма распределенного взаимного исключения с центральным координатором.  
   1. Модель с общей памятью:

#define N 5

byte enter[N]; /\* Флаги для запросов на вход в КС \*/

byte ok[N]; /\* Флаги-разрешения на вход в КС \*/

active [N] proctype P()

{

do

:: skip;

entry:

enter[\_pid] = true;

ok[\_pid] -> /\*ожидаем разрешения на вход\*/

crit: skip;

exit:

ok[\_pid] = false; /\*выход из КС\*/

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

do

:: enter[0] -> enter[0] = false; ok[0] = true;

:: enter[1] -> enter[1] = false; ok[1] = true;

:: enter[2] -> enter[2] = false; ok[2] = true;

:: enter[3] -> enter[3] = false; ok[3] = true;

:: enter[4] -> enter[4] = false; ok[4] = true;

od

/\* координатор ожидает запрос от процесса,  
затем убирает флаг запроса, и поднимает флаг на вход процесса в КС \*/

}

* 1. Модель с каналами:

#define N 5

/\* описание каналов, сообщения состоят из одного байтового поля, а сами канал может хранить N=5 сообщений одновременно \*/

chan message = [N] of {byte};/\* ид процессов, ожидающих вход\*/

chan in\_mes[N] = [1] of {byte}; /\* каналы процессов \*/

chan is\_crit\_empty = [1] of {bit};

bit is\_enable;

active [N] proctype P()

{

do

:: skip;

entry:

message ! \_pid; /\* отправка сообщения с ид процесса, выполняется если канал не переполнен\*/

in\_mes[\_pid] ? is\_enable -> /\* прием сообщения по каналу процесса с идентификатором \_pid и сохранения рез-та в переменную is\_enable \*/

crit: skip;

exit:

is\_crit\_empty ! true; ./\* отправка сообщения о том, что КС пуста \*/

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

is\_crit\_empty ! true;

byte reply;

end:

do

:: is\_crit\_empty ? is\_enable ->

if

:: message ? reply; /\* записываем номер ожидающего процесса \*/

:: timeout -> skip; /\* чтобы процесс Coordinator не работал бесконечно \*/

fi

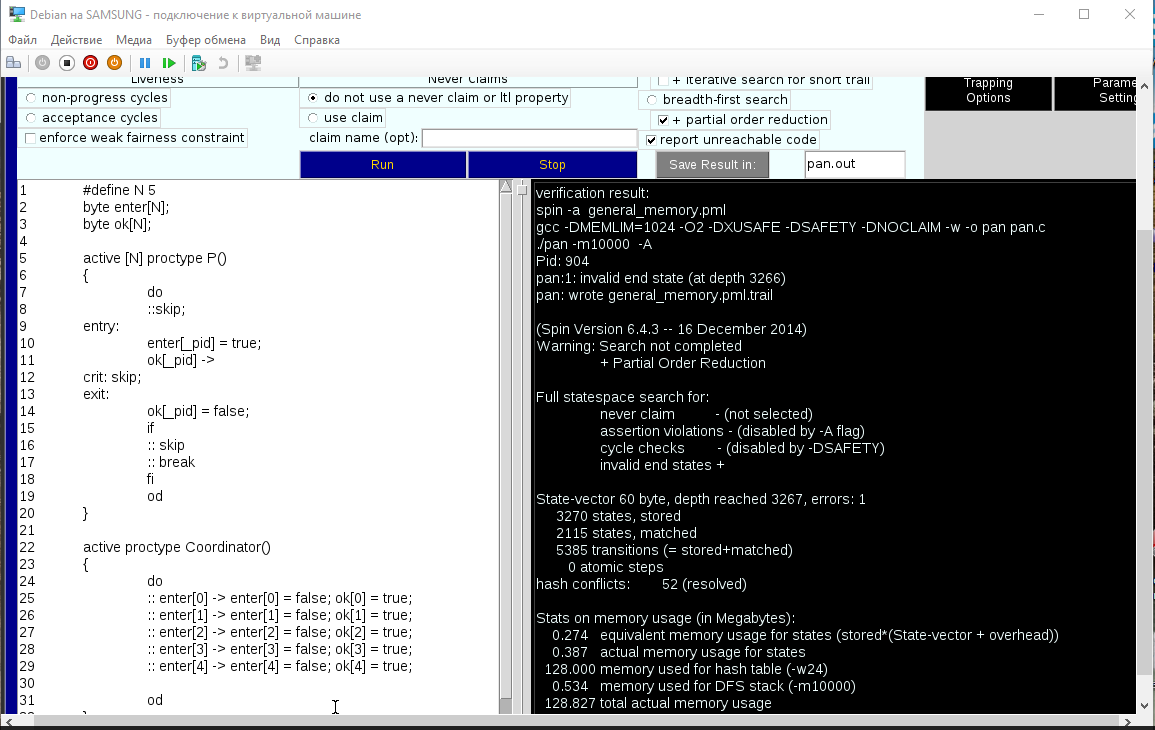
in\_mes[reply] ! true; ./\* отправка сообщения о том что процесс с идентификатором reply прошел в КС \*/

:: timeout -> break;

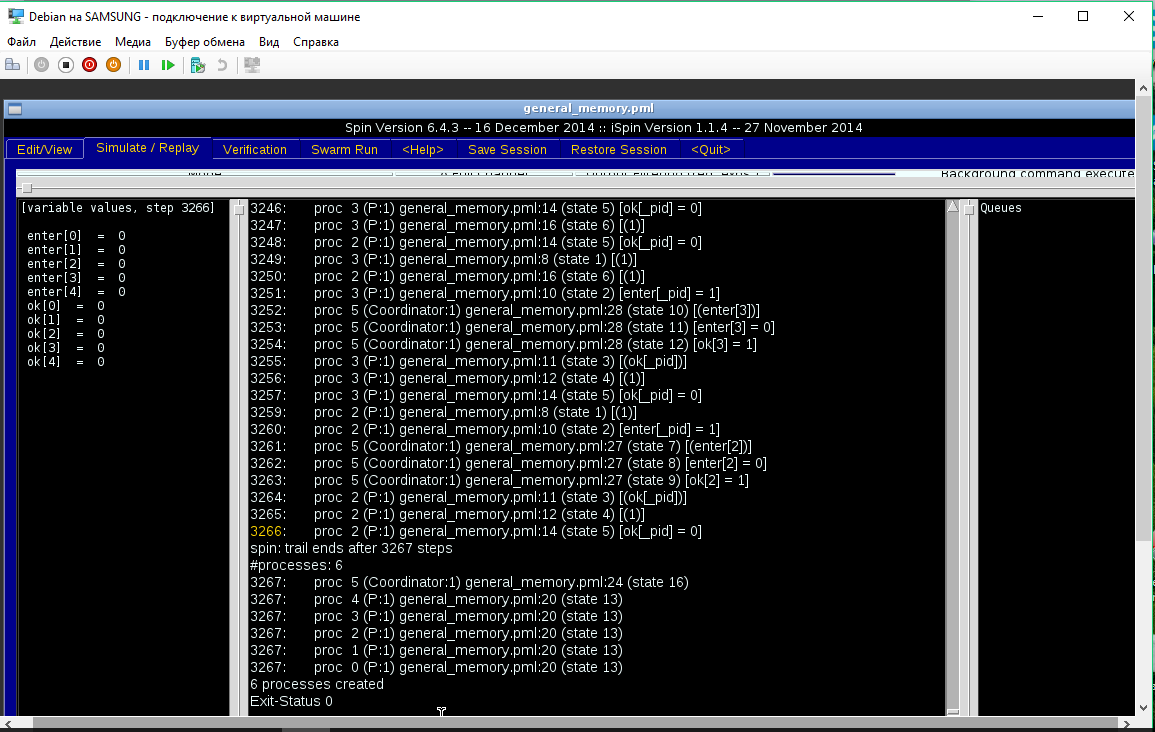
od

}

1. Верификация и исследование базовых свойств разработанных моделей.
   1. Модель с общей памятью.
      1. Свойство безопасности: свобода от взаимоблокировок.



Результат симуляции.



На 3266 шаге (at depth 3266) произошла ошибка: процесс 2 (proc 2) попытался войти в критическую секцию, уже находясь в ней. Исправить модель можно заменой повторений в Coordinator-е на цикл.

active proctype Coordinator()

{

byte i=0;

do

::

if

:: i<N ->

if

:: enter[i] -> enter[i]=false; ok[i]=true;

:: else skip;

fi

i++;

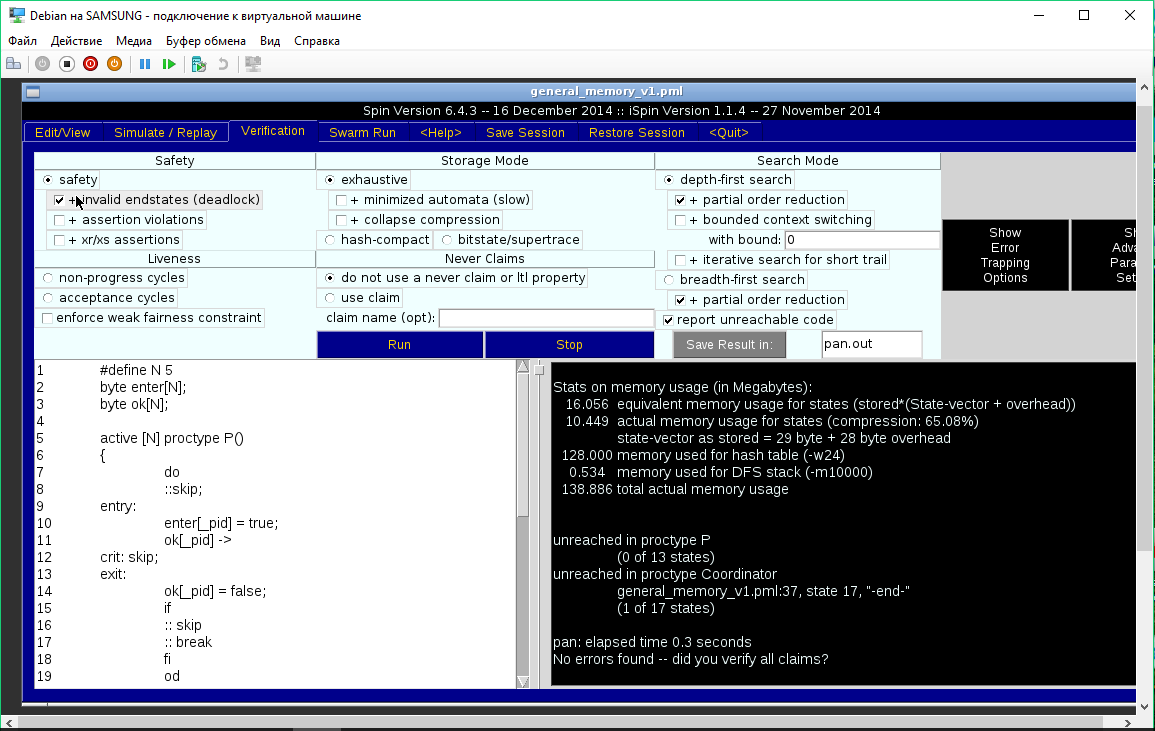
:: i>=N -> i=0; /\* обнуление итератора, чтобы цикл не был бесконечным\*/

fi

od

}

Результат верификации.



* + 1. Свойство безопасности: взаимное исключение.
       1. Локальный инвариант.

#define N 5

byte enter[N];

byte ok[N];

byte cnt; /\* счетчик процессов в КС\*/

active [N] proctype P()

{

do

:: skip;

entry:

enter[\_pid] = true;

ok[\_pid] ->

crit:

cnt++;

assert(cnt==1);

cnt--;

exit:

ok[\_pid] = false;

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

byte i=0;

do

::

if

:: i<N ->

if

:: enter[i] -> enter[i]=false; ok[i]=true;

:: else skip;

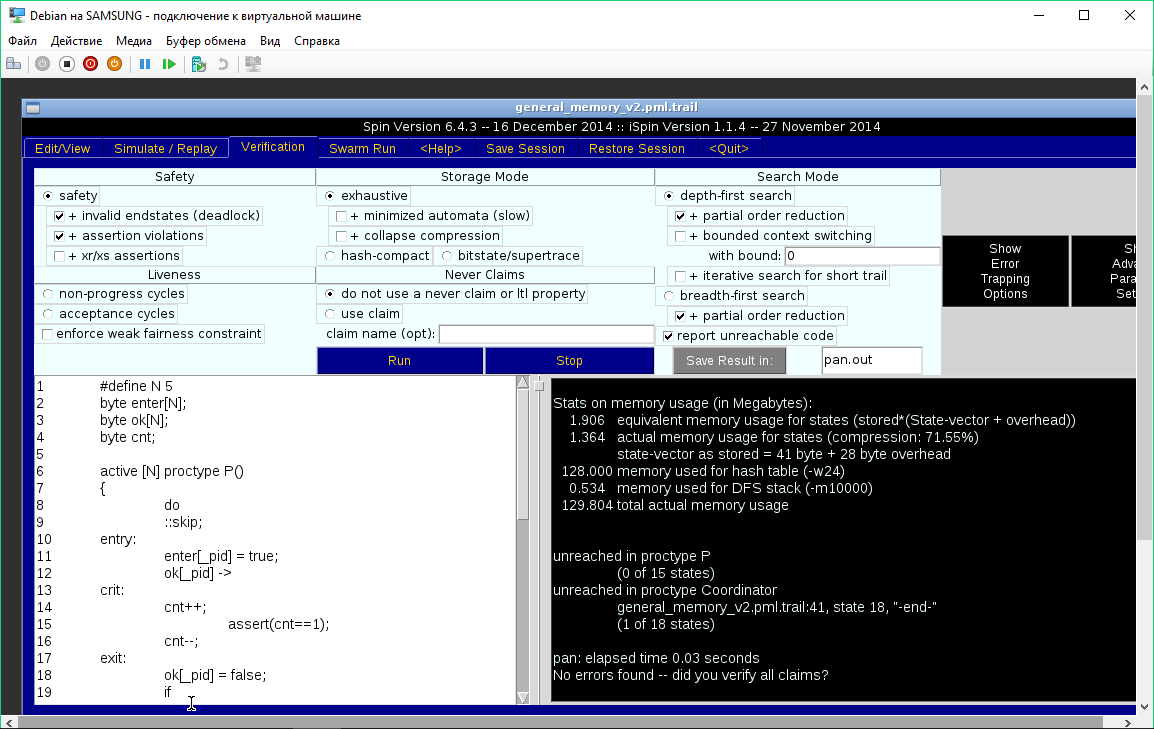
fi

!ok[i] -> i++; /\* в случае, если процесс вышел из КС увеличиваем итератор \*/

:: i>=N -> i=0;

fi

od

}

* + - 1. Глобальный инвариант.

Добавим процесс-монитор, и уберем локальный assert из КС.

#define N 5

byte enter[N];

byte ok[N];

byte cnt; /\* счетчик процессов в КС\*/

active [N] proctype P()

{

do

:: skip;

entry:

enter[\_pid] = true;

ok[\_pid] ->

crit:

cnt++;

cnt--;

exit:

ok[\_pid] = false;

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

byte i=0;

do

::

if

:: i<N ->

if

:: enter[i] -> enter[i]=false; ok[i]=true;

:: else skip;

fi

!ok[i] -> i++; /\* в случае, если процесс вышел из КС увеличиваем итератор \*/

:: i>=N -> i=0;

fi

od

}

active proctype global\_monitor()

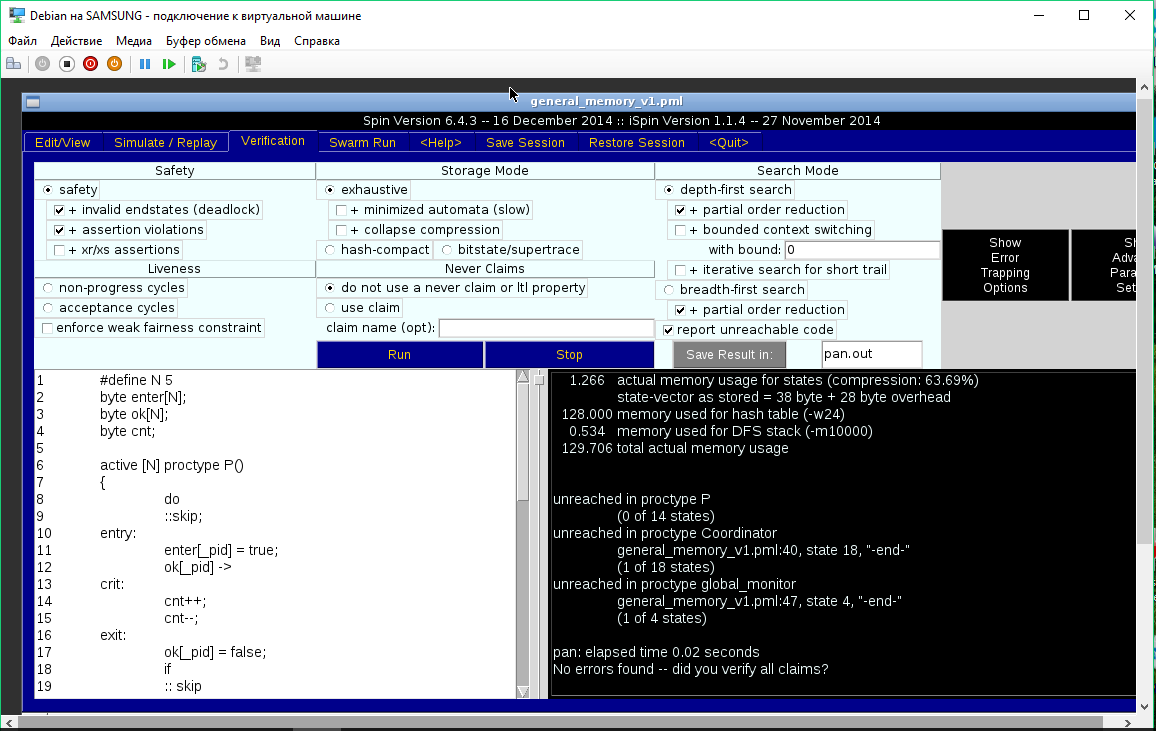
{

d\_step{/\*выполняется неделимо\*/

!(cnt<=1) -> assert(cnt<=1);

}

}



* + 1. Свойство живости.

#define N 2

byte enter[N];

byte ok[N];

byte cnt;

ltl p0 {[]([]!P[1]@entry && P[0]@entry) -> <>P[0]@crit}

**/\*** **G(G!P[1] пройдет метку entry и P[0] пройдет метку entry) -> F P[0] пройдет метку crit \*/**

ltl p1 {[](P[0]@entry -> <>P[0]@crit)}

**/\* G(P[0] пройдет метку entry) -> F P[0] пройдет метку crit \*/**

ltl p2 {[](P[0]@entry U P[1]@entry) -> [](P[0]@crit U P[1]@crit)}

**/\* G(P[0] пройдет метку entry U P[1] пройдет метку entry) -> G(P[0] пройдет метку crit U P[1] пройдет метку crit) \*/**

active [N] proctype P()

{

do

:: skip;

entry:

enter[\_pid] = true;

ok[\_pid] ->

crit:

cnt++;

assert(cnt==1);

cnt--;

exit:

ok[\_pid] = false;

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

byte i=0;

do

::

if

:: i<N ->

if

:: enter[i] -> enter[i]=false; ok[i]=true;

:: else skip;

fi

!ok[i] -> i++;

:: i>=N -> i=0;

fi

od

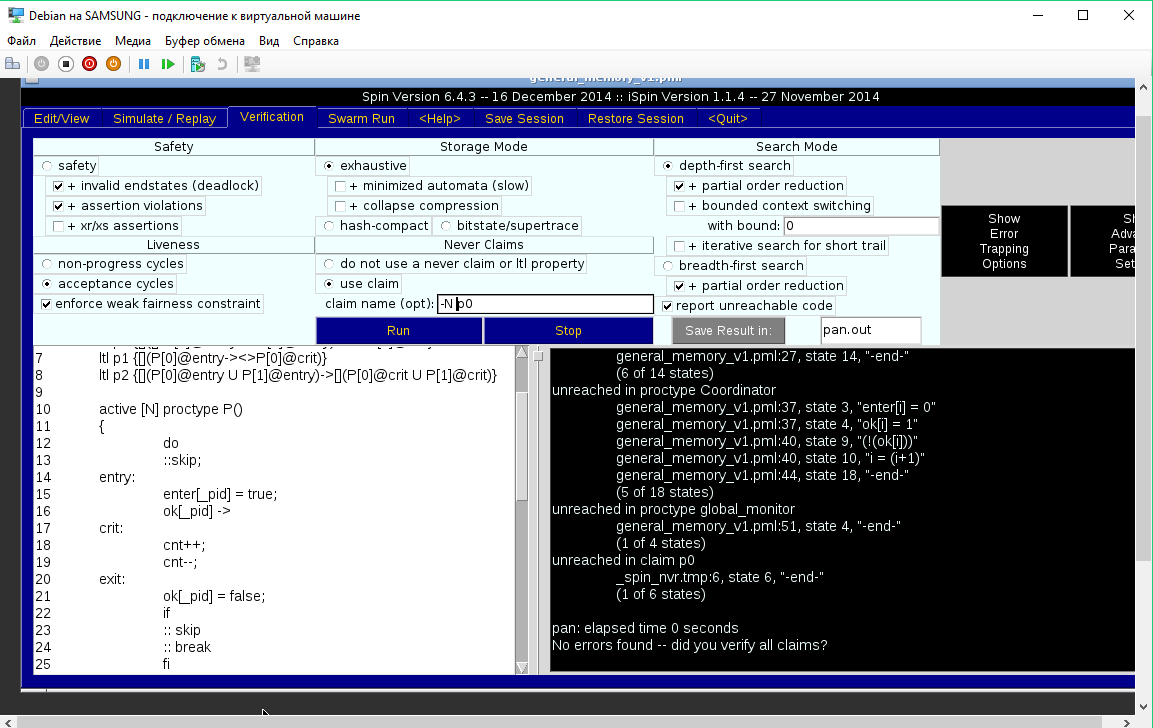
}

Первая формула проверяет, если в entry зашел p[0] и не зашел p[1], то в будущем p[0] зайдет в crit. (обязательность).

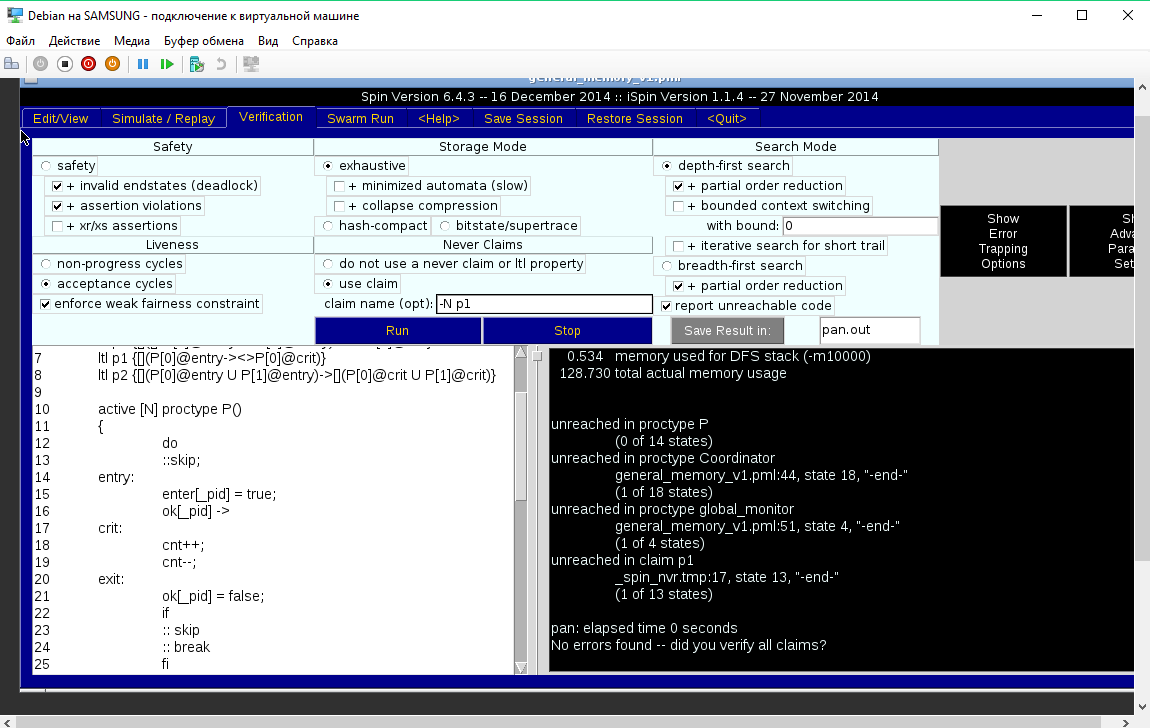
Вторая формула проверяет, если в entry зашел p[0], то в будущем он зайдет в crit. (отсутствие голодания).

Третья формула проверяет, что если p[0] зашел в entry, перед тем как p[1] зашел в entry, то p[0] заходит в crit до того, как p[1] заходит в crit. (FIFO-справедливость).

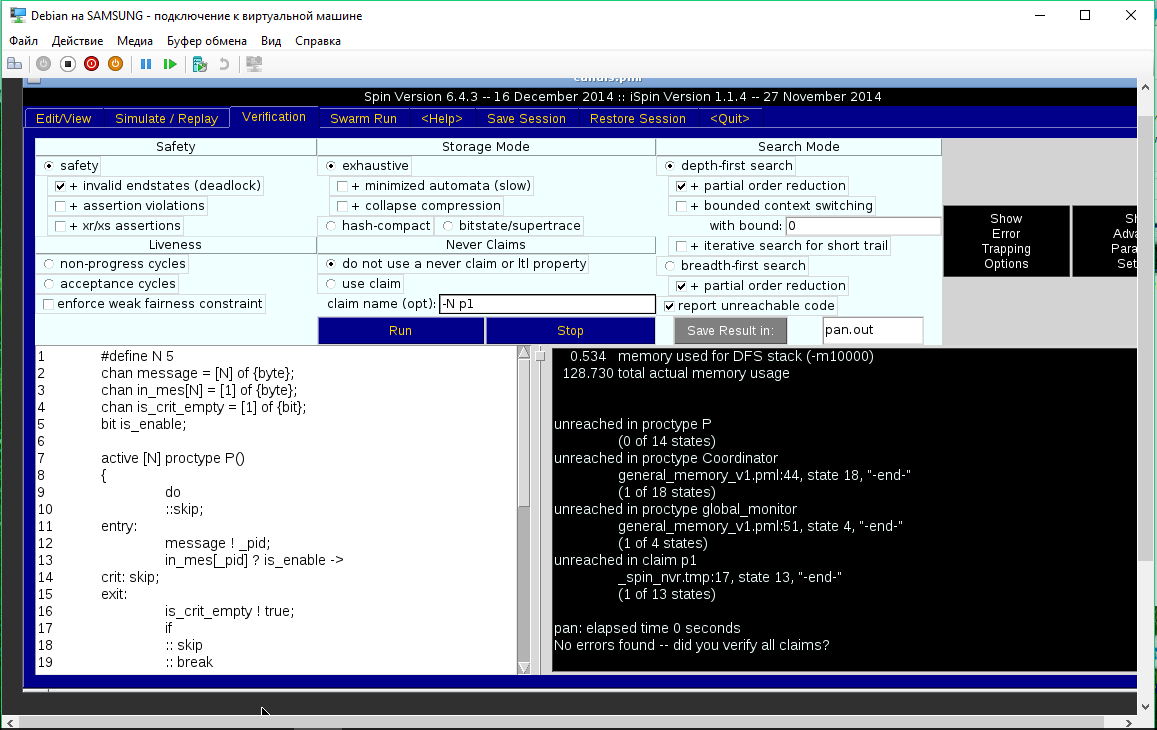
* + - 1. Свойство обязательности.



* + - 1. Свойство отсутствия голодания.



* 1. Модель с каналами.
     1. Свойство безопасности: свобода от взаимоблокировок.



* + 1. Свойство безопасности: взаимное исключение.
       1. Локальный инвариант.

каналами:

#define N 5

chan message = [N] of {byte};/\* ид процессов, ожидающих вход\*/

chan in\_mes[N] = [1] of {byte}; /\* каналы процессов \*/

chan is\_crit\_empty = [1] of {bit};

bit is\_enable;

byte cnt;

active [N] proctype P()

{

do

:: skip;

entry:

message ! \_pid; /\* отправка сообщения с ид процесса, выполняется если канал не переполнен\*/

in\_mes[\_pid] ? is\_enable -> /\* прием сообщения по каналу процесса с идентификатором \_pid и сохранения рез-та в переменную is\_enable \*/

crit: cnt++;

assert(cnt==1);

cnt--;

exit:

is\_crit\_empty ! true; ./\* отправка сообщения о том, что КС пуста \*/

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

is\_crit\_empty ! true;

byte reply;

end:

do

:: is\_crit\_empty ? is\_enable ->

if

:: message ? reply; /\* записываем номер ожидающего процесса \*/

:: timeout -> skip; /\* чтобы процесс Coordinator не работал бесконечно \*/

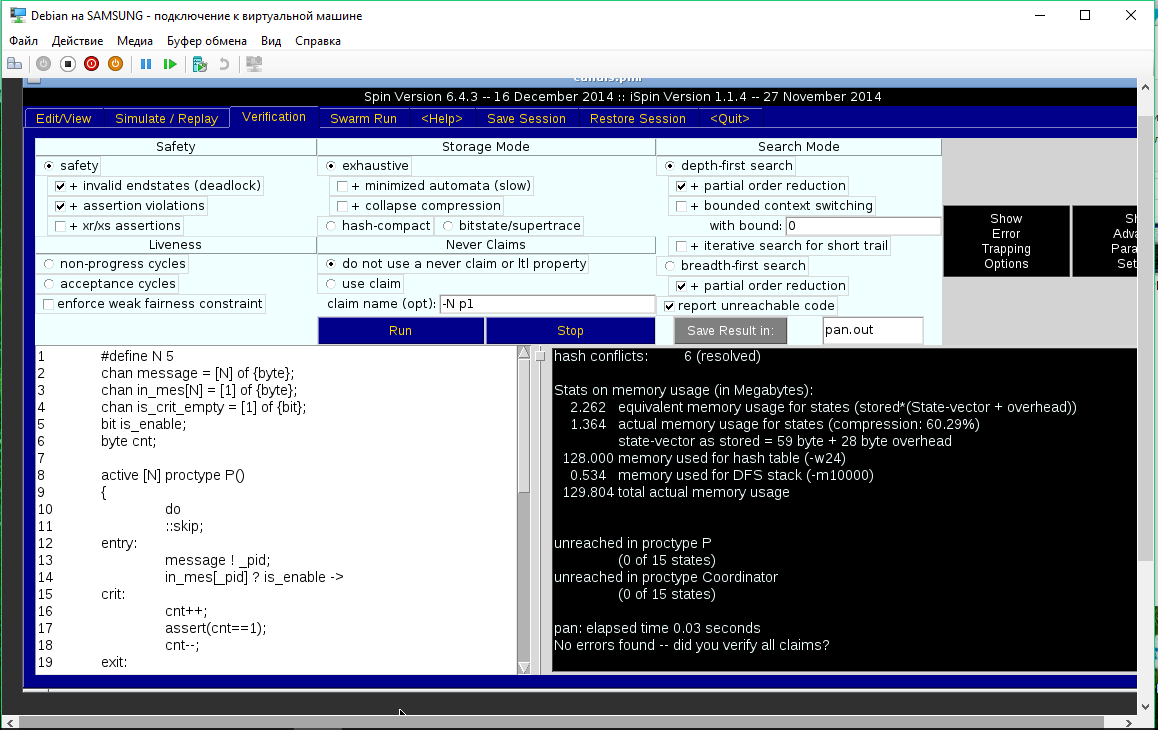
fi

in\_mes[reply] ! true; ./\* отправка сообщения о том что процесс с идентификатором reply прошел в КС \*/

:: timeout -> break;

od

}



* + - 1. Глобальный инвариант.

#define N 5

chan message = [N] of {byte};/\* ид процессов, ожидающих вход\*/

chan in\_mes[N] = [1] of {byte}; /\* каналы процессов \*/

chan is\_crit\_empty = [1] of {bit};

bit is\_enable;

byte cnt;

active [N] proctype P()

{

do

:: skip;

entry:

message ! \_pid; /\* отправка сообщения с ид процесса, выполняется если канал не переполнен\*/

in\_mes[\_pid] ? is\_enable -> /\* прием сообщения по каналу процесса с идентификатором \_pid и сохранения рез-та в переменную is\_enable \*/

crit: cnt++;

cnt--;

exit:

is\_crit\_empty ! true; ./\* отправка сообщения о том, что КС пуста \*/

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

is\_crit\_empty ! true;

byte reply;

end:

do

:: is\_crit\_empty ? is\_enable ->

if

:: message ? reply; /\* записываем номер ожидающего процесса \*/

:: timeout -> skip; /\* чтобы процесс Coordinator не работал бесконечно \*/

fi

in\_mes[reply] ! true; ./\* отправка сообщения о том что процесс с идентификатором reply прошел в КС \*/

:: timeout -> break;

od

}

active proctype global\_monitor()

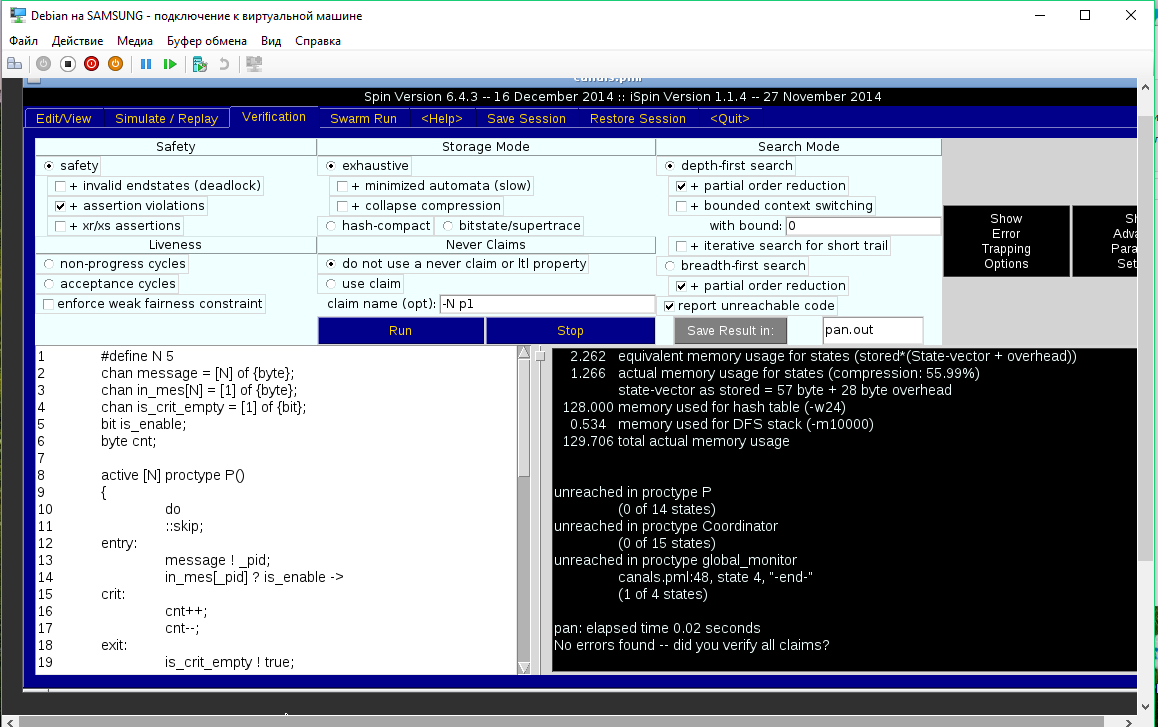
{

d\_step{/\*выполняется неделимо\*/

!(cnt<=1) -> assert(cnt<=1);

}

}



* + 1. Свойства живости.

#define N 5

/\* описание каналов, сообщения состоят из одного байтового поля, а сами канал может хранить N=5 сообщений одновременно \*/

chan message = [N] of {byte};/\* ид процессов, ожидающих вход\*/

chan in\_mes[N] = [1] of {byte}; /\* каналы процессов \*/

chan is\_crit\_empty = [1] of {bit};

bit is\_enable;

ltl p0 {[]([]!P[1]@entry && P[0]@entry) -> <>P[0]@crit}

**/\*** **G(G!P[1] пройдет метку entry и P[0] пройдет метку entry) -> F P[0] пройдет метку crit \*/**

ltl p1 {[](P[0]@entry -> <>P[0]@crit)}

**/\* G(P[0] пройдет метку entry) -> F P[0] пройдет метку crit \*/**

ltl p2 {[](P[0]@entry U P[1]@entry) -> [](P[0]@crit U P[1]@crit)}

**/\* G(P[0] пройдет метку entry U P[1] пройдет метку entry) -> G(P[0] пройдет метку crit U P[1] пройдет метку crit) \*/**

active [N] proctype P()

{

do

:: skip;

entry:

message ! \_pid; /\* отправка сообщения с ид процесса, выполняется если канал не переполнен\*/

in\_mes[\_pid] ? is\_enable -> /\* прием сообщения по каналу процесса с идентификатором \_pid и сохранения рез-та в переменную is\_enable \*/

crit: skip;

exit:

is\_crit\_empty ! true; ./\* отправка сообщения о том, что КС пуста \*/

if

:: skip

:: break

fi

od

}

active proctype Coordinator()

{

is\_crit\_empty ! true;

byte reply;

end:

do

:: is\_crit\_empty ? is\_enable ->

if

:: message ? reply; /\* записываем номер ожидающего процесса \*/

:: timeout -> skip; /\* чтобы процесс Coordinator не работал бесконечно \*/

fi

in\_mes[reply] ! true; ./\* отправка сообщения о том что процесс с идентификатором reply прошел в КС \*/

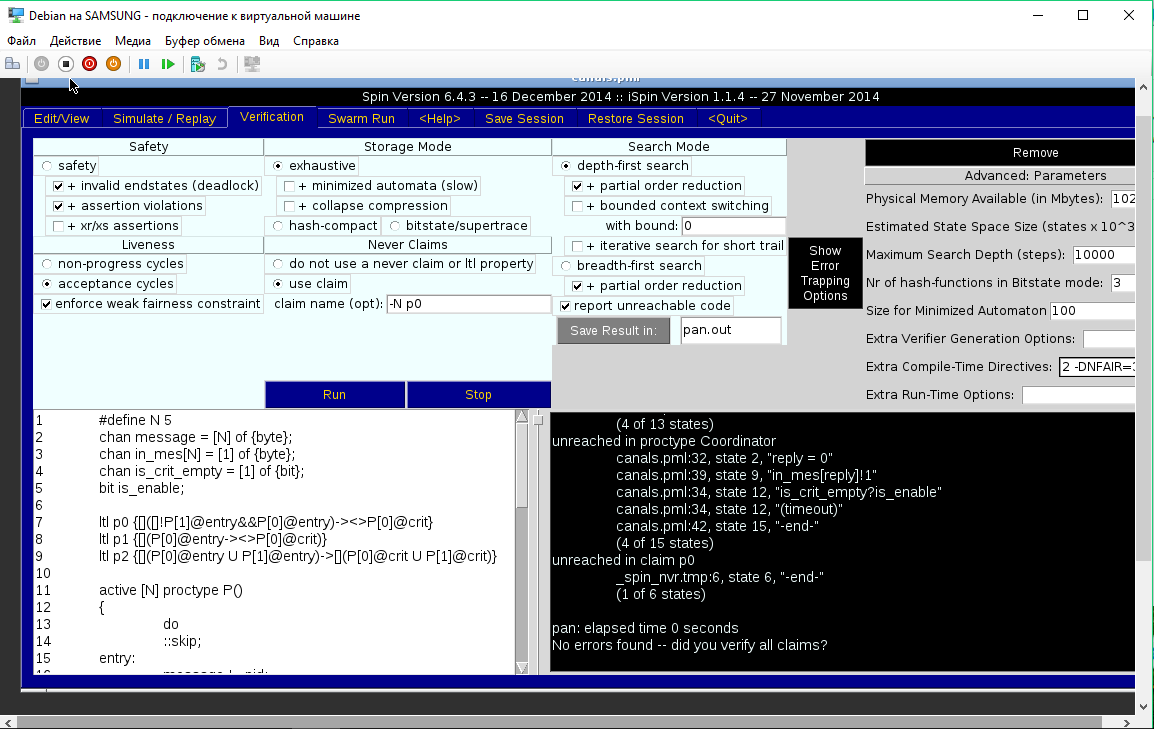
:: timeout -> break;

od

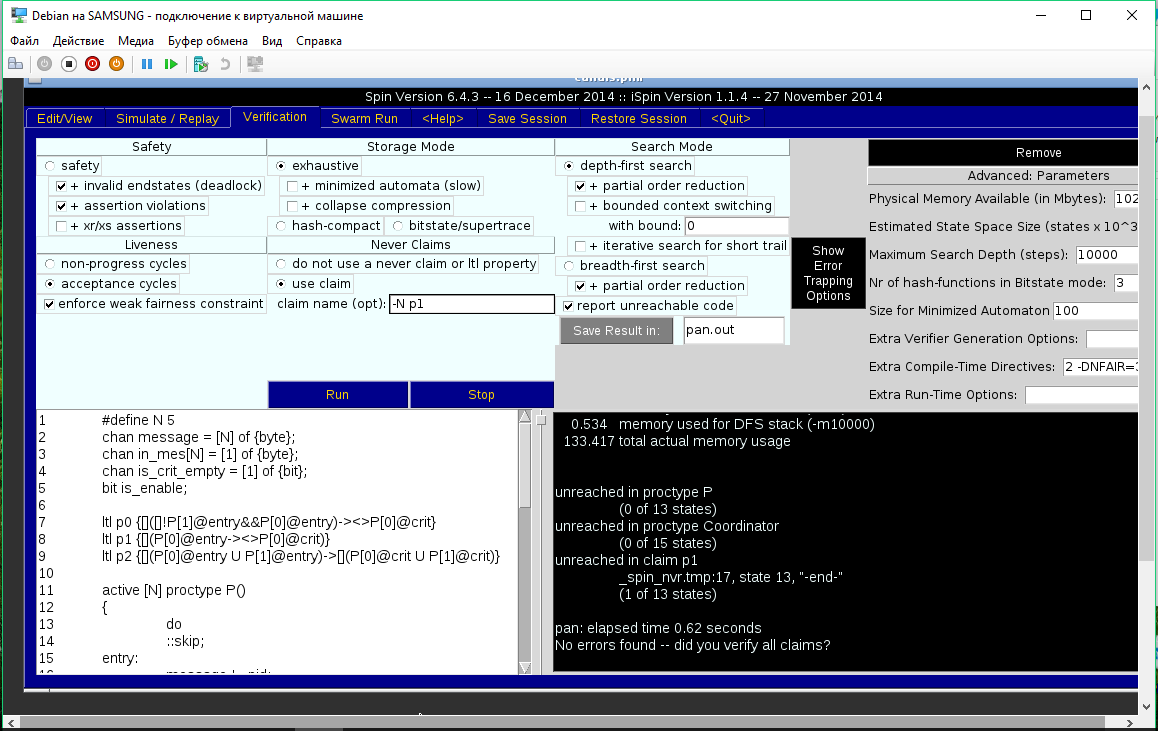
}

* + - 1. Свойство обязательности.

Для корректной работы также необходимо было включить опцию -DNFAIR=3 при компиляции кода.



* + - 1. Свойство отсутствия голодания.



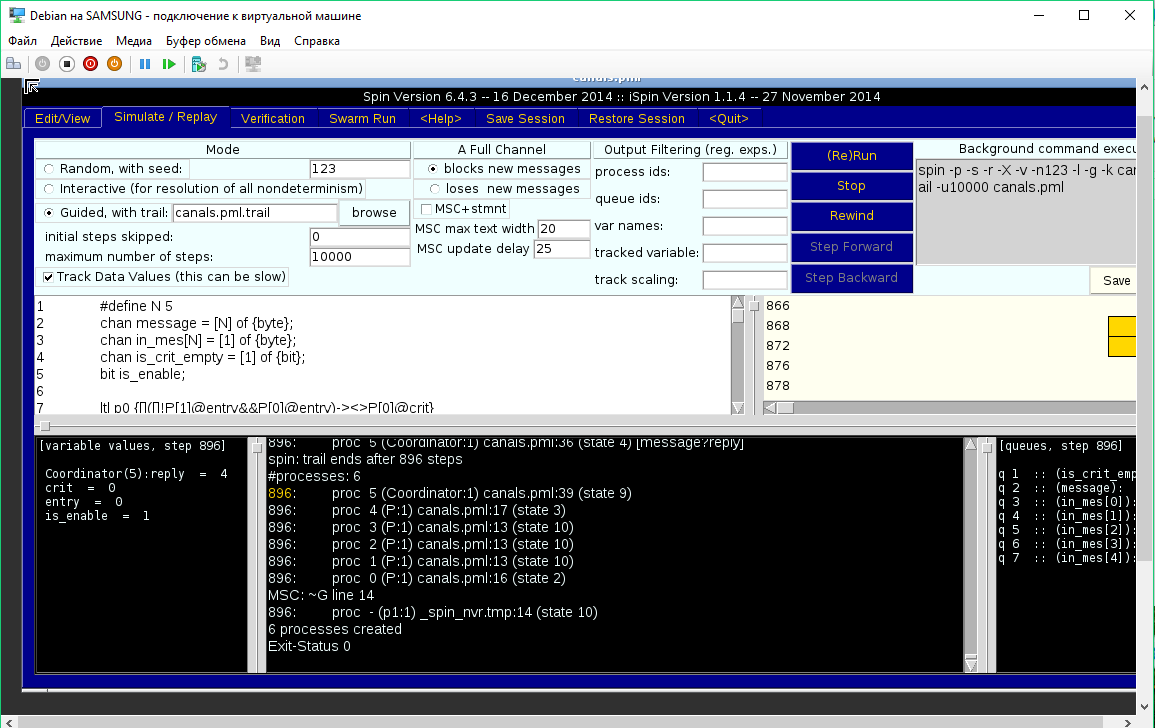
1. Ответы на вопросы:
   1. В чем недостаток централизованного алгоритма взаимного исключения?

Выход из строя процессора координатора приводит к остановке алгоритма.

* 1. В чем достоинство централизованного алгоритма взаимного исключения?

Простота реализации.

* 1. Как Вы можете объяснить результаты верификации свойства отсутствия голодания без включенной опции enforce weak fairness constraint?



Как видно из скриншота процесс с номером 0 никогда не пытается войти в критическую секцию. Чтобы это исправить необходимо в явном виде описать сильную справедливость – по Карпову «если процесс запрашивает обслуживание неопределенно часто, он получает его неопределенно часто».

* 1. Является ли разработанная Вами модель алгоритма централизованного взаимного исключения с использованием общей памяти для коммуникации FIFO-справедливой? Если нет, то каким образом ее можно сделать такой?

Разработанная модель алгоритма централизованного взаимного исключения с использованием общей памяти не является FIFO-справедливой (first in first out – первый зашел, первый вышел), потому что в ней не учитывается требование очереди: запросы на вход в КС любым процессом должны удовлетворяться в том порядке, в котором они поступили.

Чтобы сделать модель FIFO-справедливой, необходимо учитывать очередность, в которой процессы обращались к критической секции. Т.е. необходимо в отдельном глобальном массиве сохранять последовательность запросов, а по мере их выполнения в критической секции удалять первый элемент и сдвигать весь массив на единицу влево.

* 1. Является ли разработанная Вами модель алгоритма централизованного взаимного исключения с использованием каналов для коммуникации FIFO-справедливой? Если нет, то каким образом ее можно сделать такой?

Модель является FIFO-справедливой, потому что запросы на вход поступают в канал типа chan. Канала chan в Promela реализован так, что его сообщения обрабатываются по принципу FIFO.

LTL формула для свойства FIFO была описана выше (п. 2.1.3 стр.9)

**Вывод.**

В ходе выполнения данной лабораторной работы были разработаны модели алгоритма централизованного взаимного исключения с центральным координатором с использованием общей памяти и каналов. Для них проверены свойства безопасности: свобода от взаимоблокировок и взаимное исключение (с помощью локальных и глобальных инвариантов); и живучести: обязательность и отсутствие голодания. Для разработанных моделей была также осуществлена проверка FIFO-справедливости.

**Список использованной литературы.**

1. Введение в язык Promela и систему комплексной верификации Spin, И.В. Шошмина, Ю.Г. Карпов, Санкт-Петербургский государственный политехнический университет, 2009.

2. <http://spinroot.com/spin/Man/Intro.html>