Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

ОТЧЕТ ПО по Лабораторной работе №2
по курсу
«Математические основы верификации ПО»
Тема
«Моделирование гонки процессов»

Студент группы **ИУ7И-41М** **Баматраф Сохайб С.А.**

(И.О. Фамилия)

Преподаватель **Кузнецова О.В.**

(И.О. Фамилия)

***2024** г.*

Содержание

1. Введение

В этом отчете подробно описывается моделирование взаимодействия процессов с помощью программы проверки моделей SPIN. Основное внимание уделяется пониманию того, как различные методы синхронизации влияют на поведение параллельных процессов, обращающихся к общей переменной и изменяющих ее.

1.1 Цель и задачи

Цель:

Основной целью данной лабораторной работы является:

1. Продемонстрировать состояние гонки на примере простой модели Promela.
2. Применить мьютекс для разрешения состояния гонки и пронаблюдать за изменениями.
3. Использовать атомарные блоки для обеспечения корректного взаимодействия и сравнить эффективность методов синхронизации.

Задачи:

Необходимо описать взаимодействие двух процессов, работающих с одними данными. Затем место возникновения гонки необходимо дополнить мьютексами.

2. Описание модели взаимодействия процессов

Три отдельные модели Promela были использованы для моделирования взаимодействий с общей переменной:

1. RaceCondition.pml: Эта модель демонстрирует потенциальные условия гонки, когда два процесса (Writer и Reader) взаимодействуют с общей переменной без синхронизации.

2. WithMutex.pml: Включает мьютекс для управления доступом к общей переменной, чтобы предотвратить состояние гонки, наблюдаемое в первой модели.

3. Atomic_sync.pml: Использует атомарные блоки для обеспечения того, чтобы операции над общей переменной выполнялись без вмешательства параллельных процессов.

3. Демонстрация логов SPIN, в которых видна гонка

Ниже приведена версия кода Promela для файла RaceCondition.pml:

Листинг 1 — Код модели RaceCondition.pml.

```
show int shared = 1;


active proctype Writer() {
  int temp = 10;  // Assign a new value
  shared = temp;  // Write to shared variable
  printf("Writer: shared = %d\n", shared);
}
active proctype Reader() {
  int temp;
  temp = shared;  // Read from shared variable
  printf("Reader: read shared = %d\n", temp);
}
```

Для модели RaceCondition.pml выполнение SPIN привело к следующему:

Листинг 2 — Результат моделирования для RaceCondition.pml.

```
Spin Version 6.5.2 -- 6 December 2019
    + Partial Order Reduction

Full statespace search for:
    never claim             - (none specified)
    assertion violations    +
```

```
    acceptance   cycles     - (not selected)
    invalid end states      +

State-vector 32 byte, depth reached 6, errors: 0
    13 states, stored
    2 states, matched
    15 transitions (= stored+matched)
    0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
0.001       equivalent memory usage for states (stored*(State-vector + overhead))
0.292       actual memory usage for states
128.000         memory used for hash table (-w24)
0.534       memory used for DFS stack (-m10000)
128.730         total actual memory usage

unreached in proctype Writer
    (0 of 3 states)
unreached in proctype Reader
    (0 of 3 states)

pan: elapsed time 0 seconds
```

Просмотрите выходные данные модели:

Достигнута глубина: 6

Состояния: 13 сохраненных, 2 сопоставленных

Переходы: 15

Ошибки: 0

Вывод RaceCondition.pml не содержит ошибок и имеет приемлемую глубину, что указывает на то, что состояние гонки было обработано, не вызвав сбоев в системе или тупиковых ситуаций. Однако отсутствие ошибок может также означать, что потенциальное состояние гонки не проявилось таким образом, чтобы нарушить какие-либо утверждения или привести к недопустимым состояниям. Это может быть связано с тем, как SPIN планировал операции, или с отсутствием специфических условий, которые могли бы выявить состояние гонки.

4. Описание модели с мьютексом

Модель WithMutex.pml вводит мьютекс для синхронизации доступа между писателем и читателем. Такой подход позволяет эффективно управлять последовательностями доступа к общей переменной.

Ниже приведена версия кода Promela для файла WithMutex.pml:

Листинг 3 — Код модели WithMutex.pml.

```
#define P(s) atomic { s > 0 -> s-- }

#define V(s) s++

show int shared = 1;

show int semaphore = 1;  // Semaphore initialized to 1

active proctype Writer() {

  int temp = 10;
```

```
  P(semaphore);  // Wait operation on the semaphore
  shared = temp;  // Write operation protected by semaphore
  printf("Writer: shared = %d\n", shared);
  V(semaphore);  // Signal operation on the semaphore
}
active proctype Reader() {
  int temp;
  P(semaphore);  // Wait operation on the semaphore
  temp = shared;  // Read operation protected by semaphore
  printf("Reader: read shared = %d\n", temp);
  V(semaphore);  // Signal operation on the semaphore
}
```

Для модели WithMutex.pml выполнение SPIN привело к следующему:

Листинг 4 — Результат моделирования для WithMutex.pml.

```
Spin Version 6.5.2 -- 6 December 2019
    + Partial Order Reduction
Full statespace search for:
```

```
    never claim           - (none specified)
    assertion violations    +
    acceptance   cycles     - (not selected)
    invalid end states      +
State-vector 32 byte, depth reached 10, errors: 0
    19 states, stored
    1 states, matched
    20 transitions (= stored+matched)
    0 atomic steps
hash conflicts:         0 (resolved)
Stats on memory usage (in Megabytes):
0.001       equivalent memory usage for states (stored*(State-vector + overhead))
0.291       actual memory usage for states
128.000       memory used for hash table (-w24)
0.534       memory used for DFS stack (-m10000)
128.730       total actual memory usage
unreached in proctype Writer
```

(0 of 7 states)

unreached in proctype Reader

(0 of 7 states)

pan: elapsed time 0 seconds

Просмотрите выходные данные модели:

Достигнутая глубина: 10

Состояния: 19 сохранено, 1 совпало

Переходы: 20

Ошибки: 0

Результаты WithMutex.pml свидетельствуют о том, что мьютекс эффективно предотвращает любые условия гонки или ошибки. Увеличение глубины по сравнению с RaceCondition.pml отражает дополнительную сложность получения и освобождения мьютекса, но при этом достигается безошибочное выполнение и полное покрытие состояний.

5. Описание модели с атомарными

Ниже приведена версия кода Promela для файла Atomic_sync.pml:

Листинг 5 — Код модели Atomic_sync.pml.

show int shared = 1;

```
active proctype Writer() {
  int temp = 10;
  atomic {
    shared = temp;  // Atomic write operation
    printf("Writer: shared = %d\n", shared);
  }
}
active proctype Reader() {
  int temp;
  atomic {
    temp = shared;  // Atomic read operation
    printf("Reader: read shared = %d\n", temp);
  }
}
```

Для модели Atomic_sync.pml выполнение SPIN привело к следующему:

Листинг 6 — Результат моделирования для Atomic_sync.pml.

```
(Spin Version 6.5.2 -- 6 December 2019)
```

```
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 32 byte, depth reached 4, errors: 0
        7 states, stored
        1 states, matched
        8 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.292       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
unreached in proctype Writer
        (0 of 4 states)
unreached in proctype Reader
        (0 of 4 states)
pan: elapsed time 0 seconds
```

Просмотрите выходные данные модели:

Достигнутая глубина: 4

Состояния: 7 сохранено, 1 совпало

Переходы: 8

Ошибки: 0

Модель Atomic_sync.pml, использующая атомарные блоки для синхронизации, показала самое простое взаимодействие с наименьшей глубиной. Эта модель также эффективно избегает ошибок и условий гонки, как и ожидалось при использовании атомарных операций, которые по своей сути предотвращают проблемы одновременного доступа.

Заключение

Это лабораторное занятие проиллюстрировало критическую важность синхронизации в параллельных процессах. Сравнение между несинхронизированным доступом, синхронизацией на основе мьютексов и атомарными операциями позволило получить четкое представление о различных стратегиях управления доступом к общим ресурсам. Отсутствие ошибок в более сложных моделях синхронизации говорит о том, что и мьютексы, и атомарные блоки эффективны для предотвращения условий гонки и обеспечения стабильности системы.

Литература

1. https://github.com/sohaibssb/Mathematical-basics-of-verification/tree/main/ModelingProcessRace-Lab2