|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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, искусственный интеллект и системы управления\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по Лабораторной работе №2**

**по курсу**

**«Математические основы верификации ПО»**

**Тема**

**«Моделирование гонки процессов»**

Студент \_\_\_ИУ7-41М\_\_\_\_ \_\_\_\_\_Карпухин А.С.\_\_\_\_

(Группа) (И.О.Фамилия)

Преподаватель \_\_\_\_Кузнецова О.В.\_\_\_\_

(И.О.Фамилия)

*2022 г.*

**Задание**

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

**Моделирование гонки процессов**

Модель, демонстрирующая пример возникновения состояния гонки двух процессов, приведена в листинге 1.

Два одновременно запускаемых процесса копируют содержимое глобальной переменной и изменяют его с последующей перезаписью исходного значения.

Листинг 1 — Моделирование гонки процессов.

|  |
| --- |
| show int shared = 1;  active proctype Inc()  {  show int temp;  temp = shared;  temp++;  shared = temp;  }  active proctype Dec()  {  show int temp;  temp = shared;  temp--;  shared = temp;  } |

Результат работы модели приведен в листинге 2.

Листинг 2 — Результат моделирования.

|  |
| --- |
| 0: proc - (:root:) creates proc 0 (Inc)  0: proc - (:root:) creates proc 1 (Dec)  1: proc 1 (Dec:1) main.pml:16 (state 1) [temp = shared]  2: proc 0 (Inc:1) main.pml:7 (state 1) [temp = shared]  3: proc 0 (Inc:1) main.pml:8 (state 2) [temp = (temp+1)]  4: proc 0 (Inc:1) main.pml:9 (state 3) [shared = temp]  shared = 2  5: proc 1 (Dec:1) main.pml:17 (state 2) [temp = (temp-1)]  6: proc 1 (Dec:1) main.pml:18 (state 3) [shared = temp]  shared = 0  6: proc 1 (Dec:1) terminates  6: proc 0 (Inc:1) terminates  2 processes created |

Как видно из логов, оба процесса одновременно копируют одно и то же начальное значение переменной и затем независимо изменяют его. В результате разделяемая переменная сначала принимает значение 1 + 1 = 2, а затем 1 — 1 = 0.

**Использование мьютекса**

Для обеспечения исключительного доступа к разделяемым данным можно использовать механизм взаимного исключения на основе мьютекса. Пример реализации такого подхода приведен в листинге 3.

Листинг 3 — Моделирование разделяемого доступа с использованием мьютекса.

|  |
| --- |
| show int shared = 1;  show byte mutex;  #define spin\_lock(mutex) \  do \  :: 1 -> atomic { \  if \  :: mutex == 0 -> \  mutex = 1; \  break \  :: else -> skip \  fi \  } \  od  #define spin\_unlock(mutex) \  mutex = 0  active proctype Inc()  {  show int temp;  spin\_lock(mutex);  temp = shared;  temp++;  shared = temp;  spin\_unlock(mutex);  }  active proctype Dec()  {  show int temp;  spin\_lock(mutex);  temp = shared;  temp--;  shared = temp;  spin\_unlock(mutex);  } |

Мьютекс в данном примере реализован с помощью глобальной переменной mutex и ключевого слова atomic, которое позволяет сделать операции изменения переменной неделимыми. В случае, когда мьютекс занят (mutex == 1), попытка вызвать spin\_lock приведет к ожиданию его освобождения. Результат работы такой программы приведен в листинге 4. Как видно, при попытке декремента разделяемой переменной, изменяемой процессом Inc, процесс Dec блокируется в ожидании освобождения мьютекса.

Листинг 4 — Результат работы программы со spin\_lock.

|  |
| --- |
| 0: proc - (:root:) creates proc 0 (Inc)  0: proc - (:root:) creates proc 1 (Dec)  1: proc 0 (Inc:1) mutex.pml:23 (state 1) [(1)]  2: proc 1 (Dec:1) mutex.pml:34 (state 1) [(1)]  3: proc 1 (Dec:1) mutex.pml:34 (state 2) [((mutex==0))]  4: proc 1 (Dec:1) mutex.pml:34 (state 3) [mutex = 1]  mutex = 1  5: proc 1 (Dec:1) mutex.pml:34 (state 12) [break]  6: proc 0 (Inc:1) mutex.pml:23 (state 5) [else]  7: proc 0 (Inc:1) mutex.pml:23 (state 6) [(1)]  8: proc 0 (Inc:1) mutex.pml:23 (state 1) [(1)]  9: proc 0 (Inc:1) mutex.pml:23 (state 5) [else]  10: proc 0 (Inc:1) mutex.pml:23 (state 6) [(1)]  11: proc 0 (Inc:1) mutex.pml:23 (state 1) [(1)]  12: proc 1 (Dec:1) mutex.pml:35 (state 13) [temp = shared]  13: proc 1 (Dec:1) mutex.pml:36 (state 14) [temp = (temp-1)]  14: proc 0 (Inc:1) mutex.pml:23 (state 5) [else]  15: proc 0 (Inc:1) mutex.pml:23 (state 6) [(1)]  16: proc 0 (Inc:1) mutex.pml:23 (state 1) [(1)]  17: proc 0 (Inc:1) mutex.pml:23 (state 5) [else]  18: proc 0 (Inc:1) mutex.pml:23 (state 6) [(1)]  19: proc 0 (Inc:1) mutex.pml:23 (state 1) [(1)]  20: proc 0 (Inc:1) mutex.pml:23 (state 5) [else]  21: proc 0 (Inc:1) mutex.pml:23 (state 6) [(1)]  22: proc 1 (Dec:1) mutex.pml:37 (state 15) [shared = temp]  shared = 0  23: proc 1 (Dec:1) mutex.pml:38 (state 16) [mutex = 0]  mutex = 0  24: proc 0 (Inc:1) mutex.pml:23 (state 1) [(1)]  25: proc 0 (Inc:1) mutex.pml:23 (state 2) [((mutex==0))]  26: proc 0 (Inc:1) mutex.pml:23 (state 3) [mutex = 1]  mutex = 1  27: proc 0 (Inc:1) mutex.pml:23 (state 12) [break]  27: proc 1 (Dec:1) terminates  28: proc 0 (Inc:1) mutex.pml:24 (state 13) [temp = shared]  29: proc 0 (Inc:1) mutex.pml:25 (state 14) [temp = (temp+1)]  30: proc 0 (Inc:1) mutex.pml:26 (state 15) [shared = temp]  shared = 1  31: proc 0 (Inc:1) mutex.pml:27 (state 16) [mutex = 0]  mutex = 0  31: proc 0 (Inc:1) terminates  2 processes created |

Другой способ обеспечить взаимное исключение процессов — использовать atomic непосредственно с операциями над разделяемыми данными, как показано в листинге 5. Результат работы программы приведен в листинге 6. В данном случае разделяемая переменная изменяется последовательно также, как и в предыдущем варианте, но без использования дополнительной глобальной переменной mutex.

Листинг 5 — Изменение разделяемой переменной с помощью atomic.

|  |
| --- |
| show int shared = 1;  active proctype Inc()  {  show int temp;  atomic {  temp = shared;  temp++;  shared = temp;  }  }  active proctype Dec()  {  show int temp;  atomic {  temp = shared;  temp--;  shared = temp;  }  } |

Листинг 6 — Результат работы программы с использованием atomic.

|  |
| --- |
| 0: proc - (:root:) creates proc 0 (Inc)  0: proc - (:root:) creates proc 1 (Dec)  1: proc 1 (Dec:1) atomic.pml:19 (state 1) [temp = shared]  2: proc 1 (Dec:1) atomic.pml:20 (state 2) [temp = (temp-1)]  3: proc 1 (Dec:1) atomic.pml:21 (state 3) [shared = temp]  shared = 0  4: proc 0 (Inc:1) atomic.pml:8 (state 1) [temp = shared]  5: proc 0 (Inc:1) atomic.pml:9 (state 2) [temp = (temp+1)]  6: proc 0 (Inc:1) atomic.pml:10 (state 3) [shared = temp]  shared = 1  6: proc 1 (Dec:1) terminates  6: proc 0 (Inc:1) terminates  2 processes created |