|  |  |
| --- | --- |
| 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 и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № \_\_**3**\_\_**

**Дисциплина Математические основы верификации ПО**

|  |  |
| --- | --- |
| **Тема Моделирование сетевого протокола**  **Студент \_Брянская Е.В.\_\_\_\_\_\_\_\_**  **Группа \_ИУ7-41М\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_Кузнецова О.В.** |  |

Москва

2024 г

**Цель:** описать упрощённую модель сетевого протокола.

Задание: выбрать любой сетевой протокол и описать упрощённую модель этого протокола. Описать протокол и принятые допущения, привести uml-sequence, модель протокола, логи SPIN, демонстрирующие отправку/получение данных.

В качестве реализуемого протокола был выбран протокол чередующихся битов (Alternating bit protocol) – сетевой протокол канального уровня, повторно передающий потерянные или повреждённые сообщения по принципу FIFO.

Каждое сообщение от отправителя к получателю содержит данные и однобитовый порядковый номер – квитанцию, принимающий значение 0 или 1.

В случае ошибки передачи данных, отправитель повторно отправляет сообщение с теми же данными и квитанцией до тех пор, пока процесс не завершится успехом.

В случае успешного получения сообщения, получатель отправляет ответ, содержащий квитанцию с тем же битовым значением, которое было указано во входном сообщении. После того, как отправитель получает его, бит квитанции инвертируется и отправляется следующее сообщение.

Схематично это может представить в виде диаграммы:

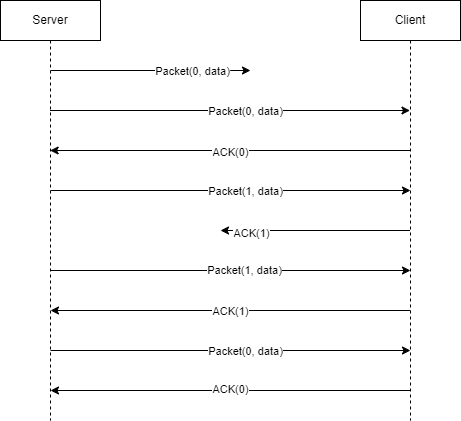


Рисунок 1 - Uml-sequence протокола

Фрагмент кода

|  |
| --- |
| typedef packet {  bit a\_bit  int data  }  int cnt = 0  chan server\_to\_client = [5] of {packet}  chan client\_to\_server = [5] of {bit}  proctype Client() {  bit a\_bit = 0  packet p  do  :: server\_to\_client?p ->  printf("[RECEIVER] Data was got, a\_bit=%d data=%d\n", p.a\_bit, p.data)  if  :: p.a\_bit == a\_bit -> client\_to\_server!p.a\_bit  a\_bit = 1 - a\_bit  :: else -> skip  fi  od  }  proctype Server() {  bit a\_bit = 0  bit ack  packet p  do  :: atomic {  p.a\_bit = a\_bit  p.data = cnt  server\_to\_client!p  printf("[SERVER] Data was send, a\_bit=%d data=%d\n", p.a\_bit, p.data)  client\_to\_server?ack  printf("[SERVER] Data was got, a\_bit=%d\n", a\_bit)  } ->  if  :: ack == a\_bit -> a\_bit = 1 - a\_bit  cnt = (cnt + 1) % 100  :: else -> skip  fi  od  }  init {  atomic {  run Server()  run Client()  }  } |

Пример лога:

|  |
| --- |
| $ pc\_spin651/spin.exe labs/lab03/lab03.pml  [SERVER] Data was send, a\_bit=0 data=0  [RECEIVER] Data was got, a\_bit=0 data=0  [SERVER] Data was got, a\_bit=0  [SERVER] Data was send, a\_bit=1 data=1  [RECEIVER] Data was got, a\_bit=1 data=1  [SERVER] Data was got, a\_bit=1  [SERVER] Data was send, a\_bit=0 data=2  [RECEIVER] Data was got, a\_bit=0 data=2  [SERVER] Data was got, a\_bit=0  [SERVER] Data was send, a\_bit=1 data=3  [RECEIVER] Data was got, a\_bit=1 data=3  [SERVER] Data was got, a\_bit=1  [SERVER] Data was send, a\_bit=0 data=4  [RECEIVER] Data was got, a\_bit=0 data=4  [SERVER] Data was got, a\_bit=0  [SERVER] Data was send, a\_bit=1 data=5  [RECEIVER] Data was got, a\_bit=1 data=5  [SERVER] Data was got, a\_bit=1  [SERVER] Data was send, a\_bit=0 data=6  [RECEIVER] Data was got, a\_bit=0 data=6  [SERVER] Data was got, a\_bit=0  [SERVER] Data was send, a\_bit=1 data=7  [RECEIVER] Data was got, a\_bit=1 data=7  [SERVER] Data was got, a\_bit=1  [SERVER] Data was send, a\_bit=0 data=8  [RECEIVER] Data was got, a\_bit=0 data=8  [SERVER] Data was got, a\_bit=0  [SERVER] Data was send, a\_bit=1 data=9  [RECEIVER] Data was got, a\_bit=1 data=9  [SERVER] Data was got, a\_bit=1  [SERVER] Data was send, a\_bit=0 data=10  [RECEIVER] Data was got, a\_bit=0 data=10  [SERVER] Data was got, a\_bit=0 |

Более детальный лог приложен ниже:

|  |
| --- |
| $ pc\_spin651/spin.exe -p labs/lab03/lab03.pml  0: proc - (:root:) creates proc 0 (:init:)  Starting Server with pid 1  1: proc 0 (:init::1) creates proc 1 (Server)  1: proc 0 (:init::1) labs/lab03/lab03.pml:54 (state 1) [(run Server())]  Starting Client with pid 2  2: proc 0 (:init::1) creates proc 2 (Client)  2: proc 0 (:init::1) labs/lab03/lab03.pml:55 (state 2) [(run Client())]  3: proc 1 (Server:1) labs/lab03/lab03.pml:33 (state 1) [p.a\_bit = a\_bit]  4: proc 1 (Server:1) labs/lab03/lab03.pml:34 (state 2) [p.data = cnt]  5: proc 1 (Server:1) labs/lab03/lab03.pml:36 (state 3) [server\_to\_client!p.a\_bit,p.data]  [SERVER] Data was send, a\_bit=0 data=0  6: proc 1 (Server:1) labs/lab03/lab03.pml:38 (state 4) [printf('[SERVER] Data was send, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  7: proc 2 (Client:1) labs/lab03/lab03.pml:16 (state 1) [server\_to\_client?p.a\_bit,p.data]  [RECEIVER] Data was got, a\_bit=0 data=0  8: proc 2 (Client:1) labs/lab03/lab03.pml:17 (state 2) [printf('[RECEIVER] Data was got, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  9: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 3) [((p.a\_bit==a\_bit))]  10: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 4) [client\_to\_server!p.a\_bit]  11: proc 2 (Client:1) labs/lab03/lab03.pml:21 (state 5) [a\_bit = (1-a\_bit)]  12: proc 1 (Server:1) labs/lab03/lab03.pml:40 (state 5) [client\_to\_server?ack]  [SERVER] Data was got, a\_bit=0  13: proc 1 (Server:1) labs/lab03/lab03.pml:42 (state 6) [printf('[SERVER] Data was got, a\_bit=%d\\n',a\_bit)]  14: proc 1 (Server:1) labs/lab03/lab03.pml:45 (state 8) [((ack==a\_bit))]  15: proc 2 (Client:1) labs/lab03/lab03.pml:24 (state 9) [.(goto)]  16: proc 1 (Server:1) labs/lab03/lab03.pml:45 (state 9) [a\_bit = (1-a\_bit)]  17: proc 1 (Server:1) labs/lab03/lab03.pml:46 (state 10) [cnt = ((cnt+1)%100)]  18: proc 1 (Server:1) labs/lab03/lab03.pml:49 (state 14) [.(goto)]  19: proc 1 (Server:1) labs/lab03/lab03.pml:50 (state 16) [.(goto)]  20: proc 1 (Server:1) labs/lab03/lab03.pml:33 (state 1) [p.a\_bit = a\_bit]  21: proc 1 (Server:1) labs/lab03/lab03.pml:34 (state 2) [p.data = cnt]  22: proc 1 (Server:1) labs/lab03/lab03.pml:36 (state 3) [server\_to\_client!p.a\_bit,p.data]  [SERVER] Data was send, a\_bit=1 data=1  23: proc 1 (Server:1) labs/lab03/lab03.pml:38 (state 4) [printf('[SERVER] Data was send, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  24: proc 2 (Client:1) labs/lab03/lab03.pml:25 (state 11) [.(goto)]  25: proc 2 (Client:1) labs/lab03/lab03.pml:16 (state 1) [server\_to\_client?p.a\_bit,p.data]  [RECEIVER] Data was got, a\_bit=1 data=1  26: proc 2 (Client:1) labs/lab03/lab03.pml:17 (state 2) [printf('[RECEIVER] Data was got, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  27: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 3) [((p.a\_bit==a\_bit))]  28: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 4) [client\_to\_server!p.a\_bit]  29: proc 1 (Server:1) labs/lab03/lab03.pml:40 (state 5) [client\_to\_server?ack]  [SERVER] Data was got, a\_bit=1  30: proc 1 (Server:1) labs/lab03/lab03.pml:42 (state 6) [printf('[SERVER] Data was got, a\_bit=%d\\n',a\_bit)]  31: proc 2 (Client:1) labs/lab03/lab03.pml:21 (state 5) [a\_bit = (1-a\_bit)]  32: proc 1 (Server:1) labs/lab03/lab03.pml:45 (state 8) [((ack==a\_bit))]  33: proc 2 (Client:1) labs/lab03/lab03.pml:24 (state 9) [.(goto)]  34: proc 2 (Client:1) labs/lab03/lab03.pml:25 (state 11) [.(goto)]  35: proc 1 (Server:1) labs/lab03/lab03.pml:45 (state 9) [a\_bit = (1-a\_bit)]  36: proc 1 (Server:1) labs/lab03/lab03.pml:46 (state 10) [cnt = ((cnt+1)%100)]  37: proc 1 (Server:1) labs/lab03/lab03.pml:49 (state 14) [.(goto)]  38: proc 1 (Server:1) labs/lab03/lab03.pml:50 (state 16) [.(goto)]  39: proc 1 (Server:1) labs/lab03/lab03.pml:33 (state 1) [p.a\_bit = a\_bit]  40: proc 1 (Server:1) labs/lab03/lab03.pml:34 (state 2) [p.data = cnt]  41: proc 1 (Server:1) labs/lab03/lab03.pml:36 (state 3) [server\_to\_client!p.a\_bit,p.data]  [SERVER] Data was send, a\_bit=0 data=2  42: proc 1 (Server:1) labs/lab03/lab03.pml:38 (state 4) [printf('[SERVER] Data was send, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  43: proc 2 (Client:1) labs/lab03/lab03.pml:16 (state 1) [server\_to\_client?p.a\_bit,p.data]  [RECEIVER] Data was got, a\_bit=0 data=2  44: proc 2 (Client:1) labs/lab03/lab03.pml:17 (state 2) [printf('[RECEIVER] Data was got, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  45: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 3) [((p.a\_bit==a\_bit))]  46: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 4) [client\_to\_server!p.a\_bit]  47: proc 2 (Client:1) labs/lab03/lab03.pml:21 (state 5) [a\_bit = (1-a\_bit)]  48: proc 1 (Server:1) labs/lab03/lab03.pml:40 (state 5) [client\_to\_server?ack]  [SERVER] Data was got, a\_bit=0  49: proc 1 (Server:1) labs/lab03/lab03.pml:42 (state 6) [printf('[SERVER] Data was got, a\_bit=%d\\n',a\_bit)]  50: proc 2 (Client:1) labs/lab03/lab03.pml:24 (state 9) [.(goto)]  51: proc 2 (Client:1) labs/lab03/lab03.pml:25 (state 11) [.(goto)]  52: proc 1 (Server:1) labs/lab03/lab03.pml:45 (state 8) [((ack==a\_bit))]  53: proc 1 (Server:1) labs/lab03/lab03.pml:45 (state 9) [a\_bit = (1-a\_bit)]  54: proc 1 (Server:1) labs/lab03/lab03.pml:46 (state 10) [cnt = ((cnt+1)%100)]  55: proc 1 (Server:1) labs/lab03/lab03.pml:49 (state 14) [.(goto)]  56: proc 1 (Server:1) labs/lab03/lab03.pml:50 (state 16) [.(goto)]  57: proc 1 (Server:1) labs/lab03/lab03.pml:33 (state 1) [p.a\_bit = a\_bit]  58: proc 1 (Server:1) labs/lab03/lab03.pml:34 (state 2) [p.data = cnt]  59: proc 1 (Server:1) labs/lab03/lab03.pml:36 (state 3) [server\_to\_client!p.a\_bit,p.data]  [SERVER] Data was send, a\_bit=1 data=3  60: proc 1 (Server:1) labs/lab03/lab03.pml:38 (state 4) [printf('[SERVER] Data was send, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  61: proc 2 (Client:1) labs/lab03/lab03.pml:16 (state 1) [server\_to\_client?p.a\_bit,p.data]  [RECEIVER] Data was got, a\_bit=1 data=3  62: proc 2 (Client:1) labs/lab03/lab03.pml:17 (state 2) [printf('[RECEIVER] Data was got, a\_bit=%d data=%d\\n',p.a\_bit,p.data)]  63: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 3) [((p.a\_bit==a\_bit))]  64: proc 2 (Client:1) labs/lab03/lab03.pml:20 (state 4) [client\_to\_server!p.a\_bit]  65: proc 2 (Client:1) labs/lab03/lab03.pml:21 (state 5) [a\_bit = (1-a\_bit)]  66: proc 1 (Server:1) labs/lab03/lab03.pml:40 (state 5) [client\_to\_server?ack]  [SERVER] Data was got, a\_bit=1 |

**Вывод**

В результате выполнения работы был описан протокол чередующихся битов, приведены диаграмма uml-sequence, код и логи, демонстрирующие отправку/получение пакетов данных.