|  |  |
| --- | --- |
| **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 ИУ «международных образовательных программ»

КАФЕДРА ИУ-7 «Программное обеспечение эвм и информационные технологии»

**ОТЧЕТ**

**по Лабораторной работе №3**

**по курсу**

**«Математические основы верификации ПО»**

**Тема**

**«Моделирование сетевого протокола»**

Студент \_\_\_ИУ7И-42М\_\_\_\_ \_Абди О.У.\_\_\_\_

(Группа) (И.О.Фамилия)

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

(И.О.Фамилия)

*2024г.*

**Задание**

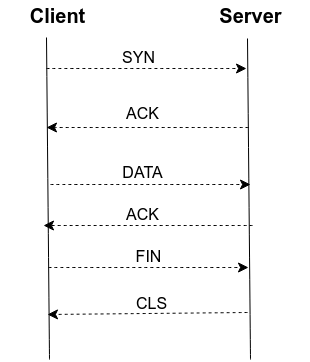
Выбирается любой сетевой протокол и описывается упрощенная модель этого протокола. Необязательно полностью все поля, например, IP-пакетов.

**Описание модели протокола**

Для реализации был выбран протокол сетевого уровня — Протокол управления передачей (TCP). Протокол был реализован в упрощенной форме с следующими предположениями:

* Клиентский и серверный процессы взаимодействуют через канал ch;
* Протокол начинается с того, что клиент отправляет сообщение SYN серверу для инициализации соединения. Получив сообщение SYN, сервер отвечает сообщением SYN\_ACK, подтверждая запрос на соединение;
* После установки соединения клиент отправляет данные (DATA) серверу, который отвечает подтверждением (ACK);
* Наконец, клиент инициирует завершение соединения, отправляя сообщение FIN, на что сервер отвечает сообщением CLS, указывая на закрытие соединения.

Диаграмма механизмов установки и закрытия соединения, реализованных в модели, приведена на рисунке 1.

Рисунок 1 - Диаграмма процесса передачи данных по протоколу.

**Листинги кода**

Код моделей отправителя (клиента) и получателя (сервера) приведен в листингах 1 и 2 соответственно.

Листинг 1 — Модель отправителя (клиента).

|  |
| --- |
| proctype client() {  int seq\_num = 0  int ack\_num = 0  int payload = 1000  // Connect to server  ch!seq\_num, SYN, payload  ch?ack\_num, SYN\_ACK, payload  seq\_num++  ack\_num++  // Send data  ch!seq\_num, DATA, payload  ch?ack\_num, ACK, payload  seq\_num++  ack\_num++  // Close connection  ch!seq\_num, FIN, payload  ch?ack\_num, CLS, payload  seq\_num++  ack\_num++  } |

Листинг 2 — Модель получателя (сервера).

|  |
| --- |
| proctype server() {  int seq\_num = 0  int ack\_num = 0  int payload = 1000  // Wait for connection request  ch?seq\_num, SYN, payload  ch!ack\_num, SYN\_ACK, payload  seq\_num++  ack\_num++  // Receive data  ch?seq\_num, DATA, payload  ch!ack\_num, ACK, payload  seq\_num++  ack\_num++  // Close connection  ch?seq\_num, FIN, payload  ch!ack\_num, CLS, payload  seq\_num++  ack\_num++  } |

Пример работы модели приведен в листинге 3.

Листинг 3 — Пример результатов работы модели.

|  |
| --- |
| 0: proc - (:root:) creates proc 0 (:init:)  Starting client with pid 1  1: proc 0 (:init::1) creates proc 1 (client)  1: proc 0 (:init::1) lab3.pml:7 (state 1) [(run client())]  Starting server with pid 2  2: proc 0 (:init::1) creates proc 2 (server)  2: proc 0 (:init::1) lab3.pml:8 (state 2) [(run server())]  3: proc 1 (client:1) lab3.pml:18 (state 1) [ch!seq\_num,SYN,payload]  4: proc 2 (server:1) lab3.pml:42 (state 1) [ch?seq\_num,SYN,payload]  5: proc 2 (server:1) lab3.pml:43 (state 2) [ch!ack\_num,SYN\_ACK,payload]  6: proc 1 (client:1) lab3.pml:19 (state 2) [ch?ack\_num,SYN\_ACK,payload]  7: proc 2 (server:1) lab3.pml:44 (state 3) [seq\_num = (seq\_num+1)]  8: proc 2 (server:1) lab3.pml:45 (state 4) [ack\_num = (ack\_num+1)]  9: proc 1 (client:1) lab3.pml:20 (state 3) [seq\_num = (seq\_num+1)]  10: proc 1 (client:1) lab3.pml:21 (state 4) [ack\_num = (ack\_num+1)]  11: proc 1 (client:1) lab3.pml:24 (state 5) [ch!seq\_num,DATA,payload]  12: proc 2 (server:1) lab3.pml:48 (state 5) [ch?seq\_num,DATA,payload]  13: proc 2 (server:1) lab3.pml:49 (state 6) [ch!ack\_num,ACK,payload]  14: proc 1 (client:1) lab3.pml:25 (state 6) [ch?ack\_num,ACK,payload]  15: proc 1 (client:1) lab3.pml:26 (state 7) [seq\_num = (seq\_num+1)]  16: proc 1 (client:1) lab3.pml:27 (state 8) [ack\_num = (ack\_num+1)]  17: proc 2 (server:1) lab3.pml:50 (state 7) [seq\_num = (seq\_num+1)]  18: proc 2 (server:1) lab3.pml:51 (state 8) [ack\_num = (ack\_num+1)]  19: proc 1 (client:1) lab3.pml:30 (state 9) [ch!seq\_num,FIN,payload]  20: proc 2 (server:1) lab3.pml:54 (state 9) [ch?seq\_num,FIN,payload]  21: proc 2 (server:1) lab3.pml:55 (state 10) [ch!ack\_num,CLS,payload]  22: proc 2 (server:1) lab3.pml:56 (state 11) [seq\_num = (seq\_num+1)]  23: proc 1 (client:1) lab3.pml:31 (state 10) [ch?ack\_num,CLS,payload]  24: proc 2 (server:1) lab3.pml:57 (state 12) [ack\_num = (ack\_num+1)]  24: proc 2 (server:1) terminates  25: proc 1 (client:1) lab3.pml:32 (state 11) [seq\_num = (seq\_num+1)]  26: proc 1 (client:1) lab3.pml:33 (state 12) [ack\_num = (ack\_num+1)]  26: proc 1 (client:1) terminates  26: proc 0 (:init::1) termin |