Отчет по лабораторной работе №3

по дисциплине:
«Математические основы верификации ПО»

Студент:	Балякин Д.В.
Группа:	ИУ7-42М
Преподаватель:	Кузнецова О.В.

Москва, 2020

Описание задания:

На языке Promela описать прототип сетевого протокола, реализующего отправление и получение данных

Код программы

```
#define CLOSED       0
#define LISTEN       1
#define SYN_SENT     2
#define SYN_RECEIVED 3
#define ESTABLISHED  4
#define FIN_WAIT_1   5
#define FIN_WAIT_2   6
#define CLOSE_WAIT   7
#define CLOSING      8
#define LAST_ACK     9
#define TIME_WAIT    10
#define EXIT         11
#define c_closed (cstate == CLOSED)
#define c_listen (cstate == LISTEN)
#define c_syn_sent (cstate == SYN_SENT)
#define c_syn_received (cstate == SYN_RECEIVED)
#define c_established (cstate == ESTABLISHED)
#define c_fin_wait_1 (cstate == FIN_WAIT_1)
#define c_fin_wait_2 (cstate == FIN_WAIT_2)
#define c_close_wait (cstate == CLOSE_WAIT)
#define c_closing (cstate == CLOSING)
#define c_last_ack (cstate == LAST_ACK)
#define c_time_wait (cstate == TIME_WAIT)
#define c_exit (cstate == EXIT)
#define s_closed (sstate == CLOSED)
#define s_listen (sstate == LISTEN)
#define s_syn_sent (sstate == SYN_SENT)
#define s_syn_received (sstate == SYN_RECEIVED)
#define s_established (sstate == ESTABLISHED)
#define s_fin_wait_1 (sstate == FIN_WAIT_1)
#define s_fin_wait_2 (sstate == FIN_WAIT_2)
#define s_close_wait (sstate == CLOSE_WAIT)
#define s_closing (sstate == CLOSING)
#define s_last_ack (sstate == LAST_ACK)
#define s_time_wait (sstate == TIME_WAIT)
#define s_exit (sstate == EXIT)
mtype = {
     SYN,
     FIN,
     ACK,
     RST,
     SYN_ACK,
     FIN_ACK,
     DATA
};
chan toclient = [1] of { mtype, int, int };
chan toserver = [1] of { mtype, int, int };
int s_do_listen = 1;
int s_do_connect = 0;
int s_do_close = 1;
int s_do_exit = 0;
int c_do_connect = 1;
int c_do_close = 0;
int c_do_exit = 0;
int c_do_rst = 0;
int c_do_send = 0;
int c_do_timeout = 1;
int sstate, cstate;
proctype Client ()
{
 mtype ctl;
 int seq = 100
 int ack = 0;
 int null;
 int msg,inseq,inack;
 int l_msg, l_seq, l_ack;
 closed:
   cstate = CLOSED;
   printf("c: closed %d\n", seq);
   if
   /* connect / SYN -> syn_sent */
   :: (c_do_connect) ->
      c_do_connect = 0; c_do_exit = 1;
      printf("c: initial connection\n");
      toserver!SYN,seq,0;
      printf("--> SYN %d %d\n", seq, ack);
      seq++;
      goto syn_sent;
   :: (c_do_exit) ->
      goto exit;
   fi;
   assert(false);              /* IMM_01 */
 listen:
   cstate = LISTEN;
   printf("c: listen %d\n", seq);
   if
   /* close / -- -> closed*/
   :: (c_do_close) ->
      goto closed;
   /* send / SYN -> syn_sent */
   :: (c_do_send) ->
      printf("--> SYN %d %d\n", seq, ack);
      toserver!SYN,seq,ack;
      seq++;
      goto syn_sent;
```

```
  fi;
  assert(false);              /* IMM_02 */
syn_sent:
  cstate = SYN_SENT;
  printf("c: syn_sent %d\n", seq);
  if
  :: toclient?msg,inseq,inack ->
     assert(inack == seq);    /* IMM_03 */
     assert(msg == SYN_ACK || msg == SYN); /* IMM_04 */
     if
     /* SYN + ACK / ACK -> established*/
     :: (msg == SYN_ACK) ->
        ack = inseq + 1;
        printf("--> ACK %d %d\n", seq, ack);
        toserver!ACK,seq,ack;
        goto established;
     /* SYN / SYN + ACK -> syn_received */
     :: (msg == SYN) ->
        ack = inseq + 1;
        printf("--> SYN_ACK %d %d\n", seq, ack);
        toserver!SYN_ACK,seq,ack;
        seq = seq + 1;
        goto syn_received;
     fi;
  :: (c_do_close) ->
     goto closed;
  fi;
  assert(false);              /* IMM_05 */

syn_received:
  cstate = SYN_RECEIVED;
  printf("c: syn_received %d\n", seq);
  if
  /* RST / -- -> listen */
  :: toclient?msg,inseq,inack ->
     assert(inack == seq);    /* IMM_06 */
     assert(msg == RST);      /* IMM_07 */
     ack = inseq + 1;
     goto listen;
  /* close / FIN -> fin_wait_1 */
  :: (c_do_close) ->
     printf("--> FIN %d %d\n", seq, ack);
     toserver!FIN,seq,ack;
     seq++;
     goto fin_wait_1;
  fi;
  assert(false);              /* IMM_08 */

established:
  cstate = ESTABLISHED;
  printf("c: established %d\n", seq);
  if
  /* close / FIN -> fin_wait_1 */
  :: (c_do_close) ->
     printf("--> FIN %d %d\n", seq, ack);
     toserver!FIN,seq,ack;
     seq++;
     goto fin_wait_1;
  /* -- / DATA */
  :: else ->
     printf ("--> DATA %d %d\n", seq, ack);
     toserver!DATA,seq,ack;
     l_msg = DATA; l_seq = seq; l_ack = ack;
     seq++;
     if
     /* ACK / -- -> established */
     :: toclient?msg,inseq,inack ->
        assert(inack == seq); /* IMM_09 */
        assert(msg == ACK);   /* IMM_10 */
        ack = inseq;
        c_do_close = 1;
        goto established;
     /* resend on timeout -> established */
     :: timeout ->
        printf("resending\n");
        toserver!l_msg,l_seq,l_ack;
        goto established;
     fi;
  fi;
  assert(false);              /* IMM_11 */

fin_wait_1:
  cstate = FIN_WAIT_1;
  printf("c: fin_wait_1 %d\n", seq);
  if
  :: toclient?msg,inseq,inack ->
     assert(inack == seq);    /* IMM_12 */
     assert(msg == ACK || msg == FIN); /* IMM_13 */
     if
     /* ACK / -- -> fin_wait_2 */
     :: (msg == ACK) ->
        ack = inseq;
        goto fin_wait_2;
     /* FIN / ACK -> closing */
     :: (msg == FIN) ->
        ack = inseq + 1;
        toserver!ACK,seq,ack;
        goto closing;
     /* FIN / FIN + ACK -> time_wait */
     :: (msg == FIN) ->
        ack = inseq + 1;
        toserver!FIN_ACK,seq,ack;
        seq++;
        goto time_wait;
     fi;
  fi;
```

```
    assert(false);              /* IMM_14 */

  fin_wait_2:
    cstate = FIN_WAIT_2;
    printf("c: fin_wait_2 %d\n", seq);
    if
    /* FIN / ACK -> time_wait */
    :: toclient?msg,inseq,inack ->
       assert(inack == seq);    /* IMM_15 */
       assert(msg == FIN);      /* IMM_16 */
       ack = inseq + 1;
       toserver!ACK,seq,ack;
       goto time_wait;
    fi;
    assert(false);              /* IMM_17 */
  close_wait:
    cstate = CLOSE_WAIT;
    printf("c: close_wait %d\n", seq);
    assert(false);              /* IMM_18 */

  closing:
    cstate = CLOSING;
    printf("c: closing %d\n", seq);
    if
    /* ACK / -- -> time_wait */
    :: toclient?msg,inseq,inack ->
       assert(inack == seq);    /* IMM_19 */
       assert(msg == ACK);      /* IMM_20 */
       goto time_wait;
    fi;
    assert(false);              /* IMM_21 */
  last_ack:
    cstate = LAST_ACK;
    printf("c: last_ack %d\n", seq);
    assert(false);              /* IMM_22 */

  time_wait:
    cstate = TIME_WAIT;
    printf("c: time_wait %d\n", seq);
    if
    /* timeout / ACK -> closed */
    :: (c_do_timeout) ->
       toserver!ACK,ack,seq;
       goto closed;
    fi;
    assert(false);              /* IMM_23 */
  exit:
    cstate = EXIT;
    printf("c: exit %d %d\n", seq, ack);
}
proctype Server ()
{
 mtype ctl;
```

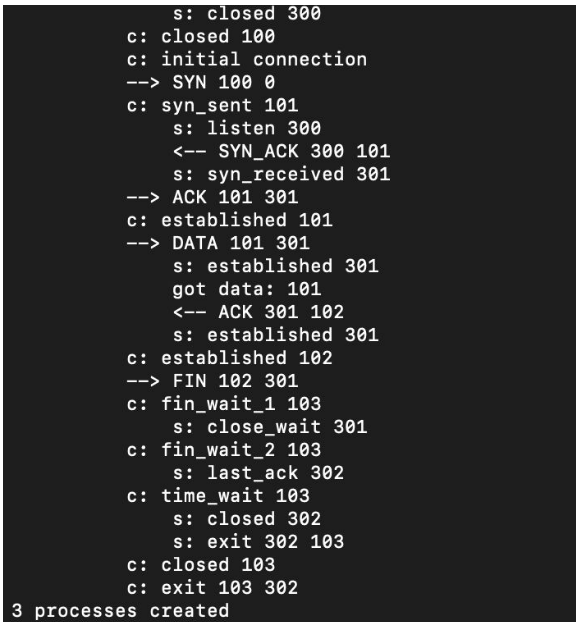
```
int seq = 300;
int ack = 0;
int null;
 int msg, inseq, inack;
int l_msg, l_seq, l_ack;
closed:
  sstate = CLOSED;
  printf("s: closed %d\n", seq);

  if
  /* listen / -- */
  :: (s_do_listen) ->
     s_do_listen = 0; s_do_exit = 1;
     goto listen;
  /* connect / SYN */
  :: (s_do_connect) ->
     s_do_connect = 0; s_do_exit = 1;
     printf("<-- %d %d\n", seq, ack);
     toclient!SYN,seq,ack;
     seq++;
     goto syn_sent;
  :: (s_do_exit) ->
     goto exit;
  fi;
  assert(false);              /* IMM_24 */
listen:
  sstate = LISTEN;
  printf("s: listen %d\n", seq);
  if
  /* SYN / SYN + ACK */
  :: toserver?msg,inseq,inack ->
     assert(msg == SYN);      /* IMM_25 */
     ack = inseq + 1;
     printf("<-- SYN_ACK %d %d\n", seq, ack);
     toclient!SYN_ACK,seq,ack;
     seq++;
     goto syn_received;
  fi;
  assert(false);              /* IMM_26 */
syn_sent:
  sstate = SYN_SENT;
  printf("s: syn_sent %d\n", seq);
  if
  /* SYN / SYN + ACK */
  :: toserver?msg,inseq,inack ->
     assert(msg == SYN);      /* IMM_27 */
     ack = inseq + 1;
     printf("<-- SYN_ACK %d %d\n", seq, ack);
     toclient!SYN_ACK,seq,ack;
     seq++;
     goto established;
  fi;
  assert(false);              /* IMM_28 */
```

```
syn_received:
  sstate = SYN_RECEIVED;
  printf("s: syn_received %d\n", seq);
  if
  /* ACK / -- */
  :: toserver?msg,inseq,inack ->
     assert(msg == ACK);      /* IMM_29 */
     assert(inack == seq);    /* IMM_30 */
     ack = inseq;
     goto established;
  fi;
  assert(false);              /* IMM_31 */

established:
  sstate = ESTABLISHED;
  printf("s: established %d\n", seq);
  if
  :: toserver?msg,inseq,inack ->
     assert(msg == FIN || msg == DATA); /* IMM_32 */
     assert(inack == seq);              /* IMM_33 */
     ack = inseq + 1;
     if
     /* FIN / ACK */
     :: (msg == FIN) ->
        toclient!ACK,seq,ack;
        goto close_wait;
     /* data / ACK */
     :: (msg == DATA) ->
        printf("got data: %d\n", ack - 1);
        printf("<-- ACK %d %d\n", seq, ack);
        toclient!ACK,seq,ack;
        goto established;
     fi;
  /* resend on timeout */
  :: timeout ->
     toserver!l_msg,l_seq,l_ack;
     goto established;
  fi;
  assert(false);              /* IMM_34 */

fin_wait_1:
  sstate = FIN_WAIT_1;
  printf("s: fin_wait_1 %d\n", seq);
  assert(false);              /* IMM_35 */

fin_wait_2:
  sstate = FIN_WAIT_2;
  printf("s: fin_wait_2 %d\n", seq);
  assert(false);              /* IMM_36 */

close_wait:
  sstate = CLOSE_WAIT;
```

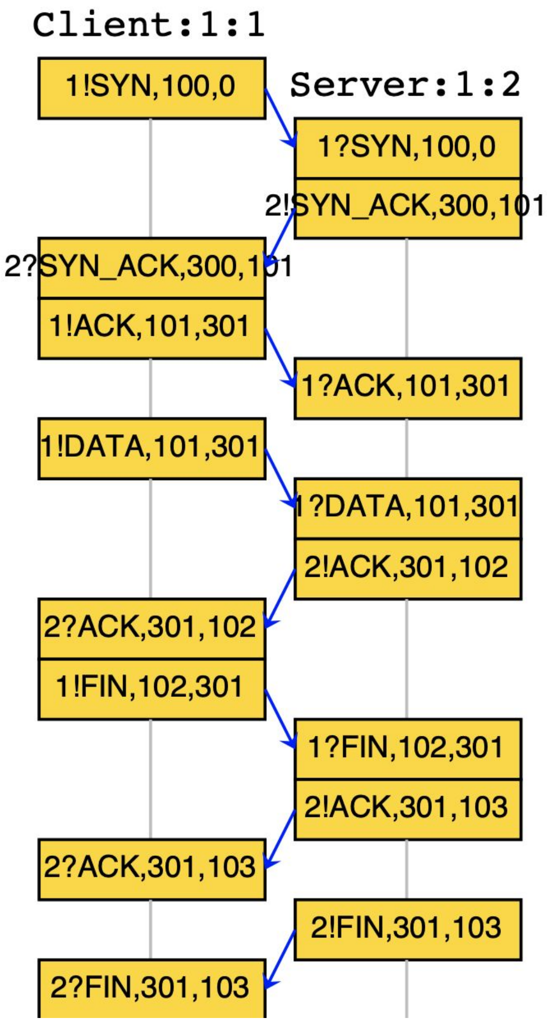
```
   printf("s: close_wait %d\n", seq);
   if
   /* close / FIN */
   :: (s_do_close) ->
      toclient!FIN,seq,ack;
      seq++;
      goto last_ack;
   fi;
   assert(false);              /* IMM_37 */

 closing:
   sstate = CLOSING;
   printf("s: closing %d\n", seq);
   assert(false);              /* IMM_38 */

 last_ack:
   sstate = LAST_ACK;
   printf("s: last_ack %d\n", seq);
   if
   /* ACK / -- */
   :: toserver?msg,inseq,inack;
      assert(msg == ACK);      /* IMM_39 */
      assert(inack == seq);    /* IMM_40 */
      ack = inseq;
      goto closed;
   fi;
   assert(false);              /* IMM_41 */

 time_wait:
   sstate = TIME_WAIT;
   printf("s: time_wait %d\n", seq);
   assert(false);              /* IMM_42 */
 exit:
   sstate = EXIT;
   printf("s: exit %d %d\n", seq, ack);
}
init
{
 run Client();
 run Server();
}
```

Код программы эмулирует установление TCP соединения и затем разрыв соединения

Выполнение программы



```
                s: closed 300
            c: closed 100
            c: initial connection
            --> SYN 100 0
            c: syn_sent 101
                s: listen 300
                <-- SYN_ACK 300 101
                s: syn_received 301
            --> ACK 101 301
            c: established 101
            --> DATA 101 301
                s: established 301
                got data: 101
                <-- ACK 301 102
                s: established 301
            c: established 102
            --> FIN 102 301
            c: fin_wait_1 103
                s: close_wait 301
            c: fin_wait_2 103
                s: last_ack 302
            c: time_wait 103
                s: closed 302
                s: exit 302 103
            c: closed 103
            c: exit 103 302
3 processes created
```



Client:1:1
Server:1:2
1!SYN,100,0
1?SYN,100,0
2!SYN_ACK,300,101
2?SYN_ACK,300,101
1!ACK,101,301
1?ACK,101,301
1!DATA,101,301
1?DATA,101,301
2!ACK,301,102
2?ACK,301,102
1!FIN,102,301
1?FIN,102,301
2!ACK,301,103
2?ACK,301,103
2!FIN,301,103
2?FIN,301,103