MODBUS TCP processing using STM32 (FREERTOS+SOCKET)

This example shows the MODBUS TCP processing using the STM32 microcontroller.
There are 6 parallel MODBUS TCP connections.
The function mbtcp_server_init() initializes the main FREERTOS thread mbtcp_server_thread. The main thread creates a new connection and a new task-handler mbtcp_serve .
If not all the connections have been established then the task-handler processes a MODBUS TCP request, creates and sends an answer, deletes itself.
If all the connections have been already established then the task-handler immediately closes the newly opened connection and deletes itself.
Thus, the new connection is established anyway. And a client program has to track a state of the connection.

```
//Init MODBUS TCP thread
void mbtcp_server_init()  {
     xTaskMBTCP = sys_thread_new("MBTCP",
                                  mbtcp_server_thread
                                  NULL,
                                  DEFAULT_THREAD_STACKSIZE * 2,
                                  MBTCPSERVER_THREAD_PRIO);
}


//Create MODBUS TCP thread
//The thread opens a new connection if
//not all connections have been established;
//increases the common connection counter
static void mbtcp_server_thread(void *arg)  {
     int sock = 0;
     int newconn = 0;
     int size = 0;
     struct sockaddr_in address;
     struct sockaddr_in remotehost;
     int res = 0;
     portBASE_TYPE xRes = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;

     //create a TCP socket
     sock = socket(AF_INET, SOCK_STREAM, 0);
     if(sock<0)  {
          printf("ERROR: MBTCP socket\n\r");
     return;
     }

     //bind to the MODBUS TCP port at any interface
     address.sin_family = AF_INET;
     address.sin_port = htons(PORT_MBTCP);
     address.sin_addr.s_addr = INADDR_ANY;

     //Socket bind
     res = bind(sock, (struct sockaddr *)&address, sizeof (address));
     if(res<0)  {
          printf("ERROR: MBTCP bind\n\r");
     return;
     }

     //listen for incoming connections (TCP listen backlog = 1)
     res = listen(sock, 1);
     if(res<0)  {
          printf("ERROR: MBTCP listen\n\r");
     return;
     }

     size = sizeof(remotehost);
     while(1)  {
          //Accept new connection
          newconn = accept(  sock,
                                  (struct sockaddr *)&remotehost,
                                  (socklen_t *)&size);
          if(newconn<0)  {
               printf("ERROR: MBTCP accept\n\r");
               return;
          }

          //If there are free connections
          if(CntrMbtcpServe<CNTR_MBTCPSERVE_MAX)  {
               //Increase connection counter
               CntrMbtcpServe++;

               //Create new serve task
               xRes = xTaskCreate( mbtcp_serve,
                                        (signed char*) "MBTCP_SRV",
                                        (4*configMINIMAL_STACK_SIZE),
                                        &newconn,
                                        MBSERVE_PRIO,
                                        NULL);
               if(xRes!=pdPASS)  {
                    printf("ERROR: MBTCP xTaskCreate\n\r");
               }
          }
          else  {
               close(newconn);
          }
     }
}


//Serve MBTCP connection
//The task serves a newly opened connection;
//decreases the common connection counter after all serving operations;
//closes the opened connection.
static void mbtcp_serve(void * pvParameters)  {
     int res = 0;

     mb_adutcp_t MbReq;
     mb_adutcp_t MbAns;
     uint16_t MbAnsLen16 = 0;

     int conn = *((int*)pvParameters);

     //------Initialize adu tcp-------
     mb_adutcp_init(&MbReq);
     mb_adutcp_init(&MbAns);

     //-------Read in the request-------
     while(1)  {
          res = read(    conn,                    //int s,
                         (void*)MbReq.Bytes,      //void *mem,
                         MB_ADUTCP_LEN_BYTES);    //size_t len)
          if(res<0)  {
               printf("ERROR: MBTCP_SERVE read\n\r");
```

```
            break;
        }
        //Connection has been closed
        if(res==0)  {
            break;
        }

        //Delay
        vTaskDelay(10);

        //-------Handle the request--------
        //Handler
        MbReqHandler(  (mb_pdu_t*) &MbReq.Pdu,
                       (mb_pdu_t*) &MbAns.Pdu,
                       (uint8_t*)  &MbAnsLen16,
                                   &mb_cl_regs,
                                   &mb_ir_regs,
                                   &mb_hr_regs);

        //Fill in Ans.Mbap
        MbAns.Mbap.Trans = MbReq.Mbap.Trans;
        MbAns.Mbap.Proto = 0x00;
        MbAns.Mbap.Len = mb_htons(MB_MBAP_UNIT_LEN_BYTES+MbAnsLen16);
        MbAns.Mbap.Unit = MbReq.Mbap.Unit;
        MbAnsLen16 += MB_MBAP_LEN_BYTES;

        //-------Send the answer--------
        res = send(    conn,                  //int s,
                       (void*)MbAns.Bytes, //const void *data,
                       MbAnsLen16,            //size_t size,
                       0);                    //int flags)
        if(res<0)  {
            printf("ERROR: MBTCP_SERVE write\n\r");
            break;
        }
        //Connection has been closed
        if(res==0)  {
            break;
        }
    }

    //Task has served connection.
    //Decrease connection counter and delete task
    CntrMbtcpServe--;

    close(conn);
    vTaskDelete(NULL);
}
```