BTN415 Term Project, Winter 2017

Milestone #3

In this milestone you will create a multi-threaded application to communicate with the robot. Using the **MySocket** you will open both a command and a telemetry socket and encapsulate the robot's application protocol defined in **PktDef** into the TCP/IP data field for transmission. A demo application is available for you to download to test your communication links (note, it only works on Windows).

Client Requirements

Your software will be the client side of the TCP/IP communications. At this point the Robot will have two listening sockets open, one for receiving operational commands and the other to transmit a telemetry stream of packets. IP Address and Port information will be provided closer to the demonstration day.

> *HINT: Your IP Address and Port numbers should be configurable and not hardcoded requiring a recompile of your client software.*

At launch, your software should set a global **bool ExeComplete** flag to **FALSE** and spawn and detach two threads from the main process. One thread for the command part of the application, and another thread to receive the telemetry stream of data. Once detached the main process should loop until **ExeComplete** is **TRUE**

Command Thread

The command thread is responsible for the command socket interface, collecting user drive information, generating packets and monitoring acknowledgements from the robot. Your command thread should contain, as a minimum, the following logic:

- Create a **MySocket** object configured as a **SocketType::CLIENT** and **ConnectionType::TCP**
- Perform the 3-way handshake to establish a reliable connection with the robots command interface
    - *NOTE: IP Address and Port information will be supplied later, but should be configurable through our software.*
- Query the user in order to get all required information to form a packet, as defined in **PktDef** (Milestone #1)
- Generate a Packet of type **PktDef** based on the user input and increment the PktCount number.
    - *HINT: Don't forget to generate the CRC*
- Transmit the Packet to the robot via the **MySocket** connection
- Wait for an acknowledgement packet from the robot
    - *REMINDER: A response packet with a command byte of zero (0) represents a Negative Acknowledgement*

This process will continue until the user requests to send a **SLEEP** command to the robot. Upon receiving an Ack packet from the robot, acknowledging the **SLEEP** command, the command thread logic should:

- Disconnect the **MySocket**
- Set the **bool ExeComplete** flag to **TRUE**
- End the thread

Telemetry Thread

The telemetry thread is responsible for the telemetry socket interface, it collects data packets that are continuously transmitted by the robot, validates the packet and parses and displays the information to the screen. Each response packet will contain a Packet Header, a Packet body with 5 bytes (shown below), and a CRC, as shown below.

Packet Header									Packet Body	Packet Trailer
PktCount	**Drive**	**Status**	**Sleep**	**Arm**	**Claw**	**Ack**	**Padding**	**Length**	**RAW Data**	**CRC**
4-bytes	1-bit	1-bit	1-bit	1-bit	1-bit	1-bit	2-bits	1-byte	5-bytes	1-byte

Each Packet Body has the following structure:

RAW Body Data							
Sonar Sensor Data	**Arm Position Data**	**Drive Flag**	**Arm Up Flag**	**Arm Down Flag**	**Claw Open Flag**	**Claw Closed Flag**	**Padding**
2-bytes	2-bytes	1-bit	1-bit	1-bit	1-bit	1-bit	3-bits

NOTE: Your code should pad out the remaining bits and set them to a default value of zero (0) in memory.

Your telemetry thread should contain, as a minimum, the following logic:

- Create a **MySocket** object configured as a **SocketType::CLIENT** and **ConnectionType::TCP**
- Perform the 3-way handshake to establish a reliable connection with the robot's command interface
- Receive and process all incoming telemetry packets from the Robot. Processing includes, but is not limited to:
    - Verification of the CRC
    - Verification of the Header data (i.e. the **STATUS** bit is set to true)
    - Display RAW data packet
    - Extract and display decimal version of the sensor data in the body of the packet
        - Sonar Reading

        - Arm Reading
    - Extract and display the Drive flag bit as 0 or 1
    - Extract and display the Arm and Claw status in English
        - Example: "Arm is Up, Claw is Closed"

NOTE: If validation fails at any point, your software should log the error to the standard out and drop the remaining processing of the packet

NOTE: IP Address and Port information will be supplied later, but should be configurable through our software.

NOTE:

There are no GUI requirements. You are free to build the user interface however you feel fit.

Simulation Software

In lieu of being allowed to test using the mobile robot, a piece of simulation software has been developed. This simulation software is written for a Windows PC and runs as a console application. It simulates the exact behavior of the software running on the mobile robot. Follow the instructions below to run the simulator with your code:

1. From the windows command prompt run ***StudentSimulator.exe***
    a. The **StudentSimulator.exe** will allow you to enter 3 command line arguments
        i. IP Address to use
        ii. Port number to use for the command interface
        iii. Port number to use for the telemetry interface
    b. By default the simulator will use an IP address of 127.0.0.1 and ports 27000 for Commands and 27501 for Telemetry
2. Start your client application and connect to the Command and Telemetry sockets
3. Send commands and evaluate the output of the telemetry you are receiving
    a. The **StudentSimulator.exe** will transmit an increasing counter in the Sonar data and a decreasing counter in the Arm data
    b. The **StudentSimulator.exe** will receive and validate all commands that are sent to it. It will inform you if the command has been accepted and then parse the command and display the "English" version of what it received