Programming Assignment 3: TCP/UDP sockets

Due Date

- See Piazza for any changes to due date and time
  - Friday ***April 27*** before midnight
  - Grading the next day
- Submit all files and directories to Perforce
  - Create a directory called: PA3 in your student directory
  - /student/<yourname>/PA3/…
    - Need to run CleanMeAll.bat *before* submission
    - Any additional files that are generated will incur point deduction
      - CleanMe.bat takes care of this
      - Needs to be modified for TCP & UDP programs
        - **4 separate solutions and code in total**

Goals

- All Required:
  - TCP socket programming
    - Learn Winsock API
    - Learn how to debug networking
  - UDP socket programming
    - In addition to the TCP problem
      - Repeat the problem for UDP
    - Note - there are UDP differences from the TCP
      - make sure you read/research carefully

Assignments

1. Use the supplied project solutions given as a starting point.
2. Write 2 TCP programs.
   a. ***Client program (using TCP protocol)***
      i. That sends individual packets of data from a list on the client to the server.
         1. There can be ***NO*** hard set number in your code, let the data determine the quantity of data. The STL vector will determine the number of elements not any predefined numbers.
            a. Any hard set numbers will lose points in grading in Client or Server.

2. Important: the packets are ***Sent Individually***
    a. Not in a collection
    b. Expecting N (example approx. 25) discrete packets, containing one value each, sent to the server
3. On the server the data will be sorted and sent back to the client.
    a. Again not a collection
    b. Expecting N (example approx. 25) discrete packets returning, one at a time, received from the server

ii. The sorted return packets will replace the existing list on the client.
iii. Output is printed before and after server sorting on the client side.

b. Server program (using TCP protocol)

i. Read and store the data given received from the client into a local list or container.
    1. Again ***NOT*** a collection
    2. Expecting N (example approx. 25) discrete packets, one at a time, received from the client
ii. Sort that data
iii. Send the sorted data ***Individually*** back to the client.
    1. Again ***NOT*** a collection
    2. Expecting N (example approx. 25) discrete packets, one at a time, Sent to the client

c. Clean and Stable

i. The client and server needs to initialize the proper network settings and establish a connection
ii. The client and server should be able to cleanly exit.
iii. The client and server should be fault tolerant
    1. If client is started before the server
        a. It should drop out gracefully and print a message

d. ***NO*** timing tweaks can be used
    i. ***No*** sleeping
    ii. ***No*** spin locks
        1. Dead (time eating) loops to deal with fragile code.
    iii. ***No*** hard coding of knowledge in the server/client
        1. Both systems need to be reactive to the data given
        2. Testing will use a different data set.
            a. "TestData.h" (Trust Me- this file will change for grading)
        3. TestData.h is only located on the client side, no including on the server side

e. API – best practice
    i. Most of the Winsock API commands return status or error codes
        1. All API's points must check return values
    ii. Your code needs to check on every API that you are not failing these errors
        1. Use Asserts() for code that's isn't critical to the execution flow

3. Submit the program in local loopback mode with port 8888
    a. Port number may have a collision
        i. Port ***8888*** should work
            1. Required for submission
    b. You can use ***localhost*** for the ip address
        i. This will allow you to local loop back
            1. Debug on ***one*** PC
4. Write additional 2 UDP programs
    a. Client program (using UDP protocol)
    b. Server program (using UDP protocol)
    c. Understand the UDP differences and mimic the same functionality of the TCP programs.
        i. Research – look at notes, books, websites
        ii. You can do it.

Validation

Simple check list to make sure that everything is checked in correctly

- TCP Client/Server program compiles and runs without crashing?
- TCP Client/Server program warning free?
- UDP Client/Server program compiles and runs without crashing?
- UDP Client/Server program warning free?
- Is the data sorted and sent individually?
- Does the application hang?

Troubleshooting

- Port number may have a collision
  - Very unlikely but if it does change it to another number.
  - Port ***8888*** should work, if not use 60607 (School's Zip code)
- You can use ***localhost*** for the ip address
  - This will allow you to local loop back (debug on one PC)
  - If you use something else
    - Please return it to ***localhost*** for submission
    - Needs to happen for grading purposes
- Open 2 instances of visual studio in 2 separate windows.
  - Add break points and toggle back and forth between games.
  - It hurts but you need to do this.

Hints

Most assignments will have hints in a section like this.

- Baby steps, use an very incremental process
  - Big steps will prevent you from finishing task
- Look at my suggestions it will save you work maybe even days of struggle
  - See Appendix
- Look up these critical references
  - Getting Started with Winsock MSDN
  - Winsock Structures MSDN
  - Winsock Functions MSDN
- ***Winsock g***etting started
  - Look at their tutorial, very useful
  - Also on that site are function descriptions and manuals

PA 3: TCP sockets - Suggestions

Attached is a flow diagram for this assignment that I used when I did it.
We are not using multi-threads or events, so you need to do a little more packet transmission.
Or at least that's what I did.

Remember no sleeping or timing loops. This program needs to be fast and deterministically executed.

I created 2 specialized packets, they were both simple struct/classes.

Transmission packet:

- *enum* (size of int): Type Identifier
- *int*: Number of Packets in the send
- *int*: Size of an individual packet

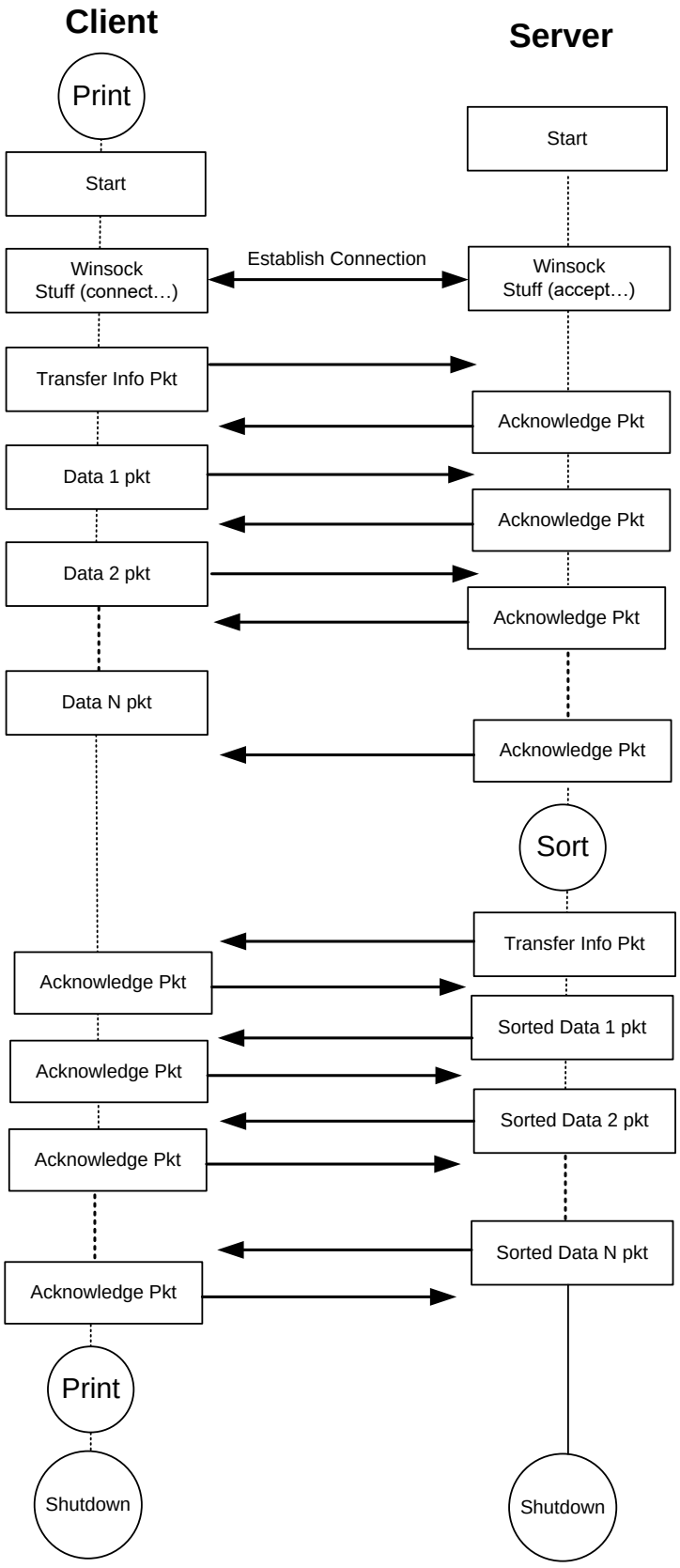
Acknowledge packet

- *int*: type identifier (to identify it seperately from the transmission packet)

I found by return the acknowledgement packet between data node deliveries, the synchronization and race conditions went away. Other paradigms always may work for you. Experiment and enjoy

See the next page

-Prof K.


Client
Server
Print
Start
Start
Winsock Stuff (connect…)
Establish Connection
Winsock Stuff (accept…)
Transfer Info Pkt
Acknowledge Pkt
Data 1 pkt
Acknowledge Pkt
Data 2 pkt
Acknowledge Pkt
Data N pkt
Acknowledge Pkt
Sort
Transfer Info Pkt
Acknowledge Pkt
Sorted Data 1 pkt
Acknowledge Pkt
Sorted Data 2 pkt
Acknowledge Pkt
Sorted Data N pkt
Acknowledge Pkt
Print
Shutdown
Shutdown