# CST 415 – Assignment 1

Wilsonville, Fall 2019

Create and test a simple port reservation service using **ONE** of the following:

1. .NET System.Net. with C# on Windows.
2. Win32 winsock2 with C++ on Windows (extra credit)
3. UNIX sockets with C++ on UNIX (extra credit)

If you choose C# .NET, you may start with the code in the Assignment 1 handout zip file.

The service will be used with other assignments this term.

The PRS executable should accept the following command line arguments:

-p <PRS port>

-s <starting client port number>

-e <ending client port number>

-t <keep alive time in seconds>

<PRS port> is the port that your PRS uses to communicate with clients.

Manage ports between the starting client and ending client numbers (inclusive).

Keep alive time specifies how long a port can remain open without a Keep Alive message.

Default values: -p 30000 –s 40000 –e 40099 –t 300

All communication with the server will use UDP.

Your program must accept and respond to the following requests (defined below):

* Request port – reserve an unused port number for a named service
* Lookup port – find the port reserved for the named service
* Keep port alive –port number is still in use and should remain reserved
* Close port – port number is no longer in use and can be made available
* Stop – close down the PRS

The following struct defines the message format for making and responding to requests:

typedef struct

{

int8\_t msg\_type; // 1 byte

char service\_name[50]; // 50 bytes

uint16\_t port; // 2 bytes

int8\_t status; // 1 byte

} request\_t; // 54 bytes total

msg\_type can have one of the following values:

REQUEST\_PORT = 0

LOOKUP\_PORT = 1

KEEP\_ALIVE = 2

CLOSE\_PORT = 3

STOP = 4

RESPONSE = 5

status can have one of the following values:

SUCCESS = 0

SERVICE\_IN\_USE = 1

SERVICE\_NOT\_FOUND = 2

ALL\_PORTS\_BUSY = 3

INVALID\_ARG = 4

UNDEFINED\_ERROR = 5

Every time the PRS receives a message, it must return a RESPONSE message, even if the received message is not well formed.

All messages (including the RESPONSE) must specify the service name as a null terminated string in service\_name. This means names must be no more than 49 chars.

The KEEP\_ALIVE, CLOSE\_PORT, and RESPONSE messages must have the port number specified in port. The port number must be in **network byte order**.

The following are the proper responses to each message type:

**REQUEST\_PORT**: Must return a RESPONSE with the lowest numbered unused port. If all ports are in use, status must be set to ALL\_PORTS\_BUSY.

**LOOKUP\_PORT**: If the named service is in the database, must return the port number for the corresponding service. Otherwise, must return a response with SERVICE\_NOT\_FOUND.

**KEEP\_ALIVE**: Must update the timeout value for the service.

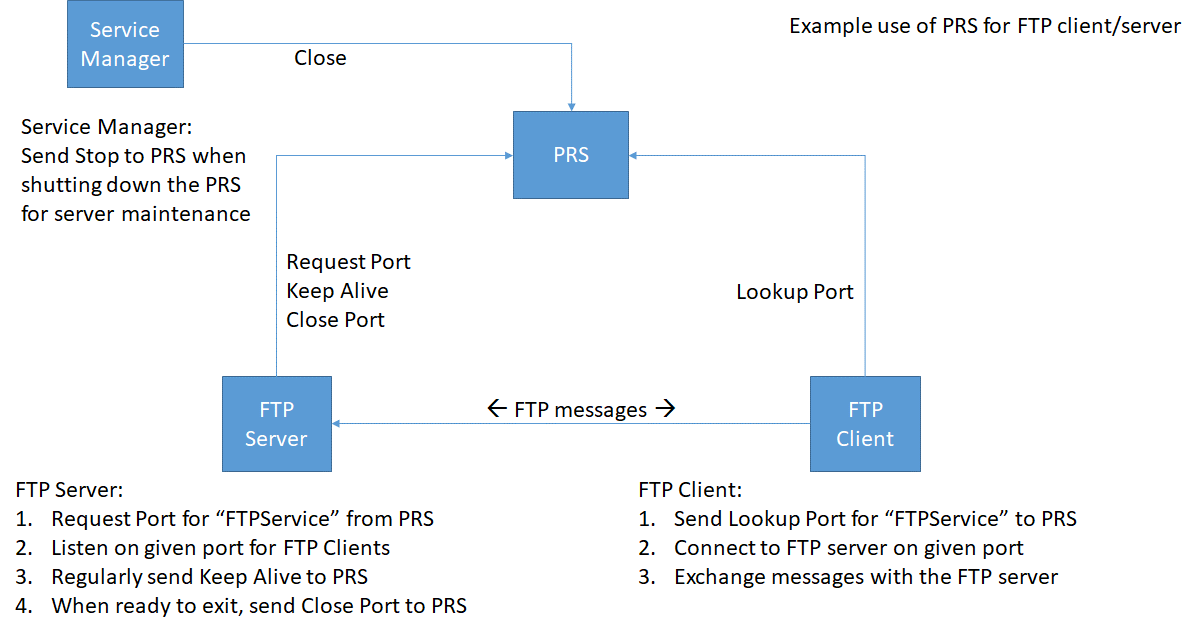
**CLOSE\_PORT**: Must remove the service from the database and make the port number available.

**STOP**: Must cause a clean shut-down of the PRS.

If a port is allocated and no KEEP\_ALIVE message is received within the keep alive time, the service using that port will be assumed to no longer need the port. It will be removed from the database as if a CLOSE\_PORT message had been received. Note: neither timers nor threads are necessary to meet this requirement. It is sufficient to check for timeouts when processing each received message. If there are no requests for an extended period of time, ports may be used well after the specified timeout value. However, if there are no requests, there is no harm in this.

Your program should always allocate the lowest numbered unused port.

Example use of the PRS by an FTP Server, FTP Client and Service Manager:



Pseudo-code for the example FTP server using the PRS:

create socket for talking to PRS

send REQUEST\_PORT message to PRS for “FTPService”

receive response message from PRS

if (response.status != SUCCESS)

{

// SERVICE\_IN\_USE, ALL\_PORTS\_BUSY,

// INVALID\_ARG, or UNDEFINED\_ERROR

exit w/ appropriate error message

}

create socket to talk with FTP clients

bind socket to response.port recevied from PRS

while (!done)

{

process requests from FTP clients

send KEEP\_ALIVE message to PRS regularly

}

send CLOSE\_PORT message to PRS

exit normally

Pseudo-code for the example FTP client using the PRS:

create socket for talking to PRS

send LOOKUP\_PORT message to PRS for “FTPService”

receive response message from PRS

if (response.status != SUCCESS)

{

// SERVICE\_NOT\_FOUND,

// INVALID\_ARG, or UNDEFINED\_ERROR

exit w/ appropriate error message

}

create socket to talk with FTP server

connect socket to response.port recevied from PRS

while (!done)

{

send FTP requests to server and process responses

}

exit normally

Pseudo-code for the exaple Service Manager managing the PRS:

if (user chooses option to stop PRS)

{

create socket for talking to PRS

send STOP message to PRS

}

Highly Recommended: Implement a test client in the same solution with your PRS. See the document “CST 415 – Assignment 1 Test Cases.docx” for example test cases that you can use in your test client. A skelleton test client is in the handout zip file.

NOTE: for testing the timeout feature, the timeout should be kept short (a few seconds).

Turn in you source code in a zip file called CST415\_Assign1\_your\_name.zip (e.g. “CST415\_Assign1\_Pete\_Myers.zip”. The instructor should be able to compile, run and verify your program. This means you need to include your Solution/Project or Makefile as appropriate.

## Handout Code

The “CST415 – Assignment 1 Handout” zip file includes the following classes. Some classes and methods are already implemented. Each method that requires implementation contains a TODO comment.

