

**The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2009**

COMS3200 / COMS7201 – Assignment One

Due: 4pm Monday September 14, 2009

Assignment weight 16%

Introduction

The aim of this assignment is for you to gain experience in both designing interprocess communication for networked (distributed) applications and in programming such applications using socket level primitives for the TCP protocol. The assignment has two parts: 1) a design of communication primitives and message formats for a given scenario of communicating processes, and 2) implementation of the designed communication using TCP sockets.

This is an **individual assignment**. You may discuss aspects of the design, programming and the assignment specification with fellow students, but should not actively help (or seek help from) other students with the actual design and coding of the assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that submitted code will be subject to checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. If you're having trouble, seek help from the tutor or the lecturer – don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school website: <http://www.itee.uq.edu.au/about ITEE/policies/student-misconduct.html>

Part A (6%)

The aim is to design a system of communicating processes using client/server and message passing. You must choose appropriate communication primitives and design suitable message formats.

The available communication primitives are assumed to be built on top of a reliable message passing transport service. Messages are of arbitrary length. The available primitives are:

- blocking send
- non-blocking send
- blocking receive
- non-blocking receive
- RPC call
- RPC server accept
- RPC reply

Requirements

There are five *types* of processes in the system. These are

- Name Server (there is only ever one process of this type)
- Stock exchange (there is only ever one process of this type)

- Share registry (there is only one process of this type)
- Broker (there may be many processes of this type)
- Investor (there may be many processes of this type)

The following requirements are to be supported

- Processes that have to be found by other processes (servers) register with the Name Server providing their current IP address and port and other processes query the Name Server for this information.
- Each investor is identified by a unique holder-identification number (HIN) which has 10 alphanumeric characters.
- Investors communicate only with brokers (and the name server).
- Investors issue requests of one of three types: quote, buy or sell. In a quote request, an investor asks the broker for the last sale price of a given stock, or for all known last sale prices. In a buy request, an investor specifies that they wish to buy a certain number of shares of a certain stock (optionally with a maximum price per share they wish to pay). In a sell request, an investor specifies that they wish to sell a certain number of shares of a certain stock (optionally with a minimum price they wish to receive per share). Investors take no further action until they receive a reply from the broker.
- Brokers respond to investor requests. For a quote request, the broker returns the last known sale price for the given stock. If no last sale price is on record for that stock, then an indication of this is returned to the investor. If no stock is specified, the broker returns all of the last sale prices it has on record (possibly 0). For buy and sell requests, the broker passes the request on to the stock exchange and returns the status of the transaction to the investor when that information is returned by the stock exchange. The broker may deal with other requests (from other investors) in the mean time. Brokers maintain a record of last sale prices for all stocks that have been sold (since the process started).
- The stock exchange receives buy and sell requests from brokers (forwarded from investors) and tries to match such requests (based on the particular stock, the number of shares to be sold/bought and any price information included in the request). Only complete parcels of shares are sold/bought, e.g. a request to buy 200 shares can be met only by selling a package of 200 shares, not by selling two packages each of 100 shares or by splitting up a larger package.
- The stock exchange will hold a request for the time specified at process run time (e.g. up to 20 seconds) in an attempt to find a matching request in order to complete the transaction. If a match occurs within this time, the stock exchange will inform the brokers of the selling/buying investors of the sale and of the agreed sale price. All brokers will be informed of the last sale price of that stock. If a transaction is not matched within the specified time of it arriving at the stock exchange, then the stock exchange notifies the broker who submitted the request that a sale was not able to be completed.
- Investors are informed by brokers of the status of any buy/sell request - i.e. they are informed of the sale price if the transaction was completed, or they are informed of the non completion if no sale took place.
- After a sale, the stock exchange will inform the share registry of the HINs of both the seller and the buyer, the names of their brokers, the stock code, and, the number of shares transferred.
- Stocks are identified by a 6 letter code.
- Brokers are identified by a string of 20 characters.
- All processes which must be “found” by another process must first register with the name server. The stock exchange is to use the name “Stock Exchange” (without the quotes); the

share registry is to use the name “Share Registry” (without the quotes). A broker may not use either of these names.

- Share prices are dollars and cents and will be in the range 0.01 to 1,000,000.00.
- The number of shares bought/sold in any one transaction will be in the range 1 to 1,000,000.
- All processes are to be runnable on a command-line (e.g. `java Registry`) and are to accept command line arguments (i.e. parameters). (For example for Java it is assumed that the Java CLASSPATH is set appropriately.) These command line arguments are to be:
 - Investor process - the Holder Identification Number (HIN) followed by the name of a broker to deal with
 - Broker - the Broker name
 - Stock Exchange - request hold time (in seconds, defaults to 20 seconds if not given)
 - Share Registry - no argument expected
- An investor process must prompt (via standard error) for user input (via standard input) as follows. It must first prompt for the transaction type - either “q” for quote, “s” for sell, or “b” for buy. It should then prompt for the stock code. If the transaction is a sell or buy transaction, it should then prompt for the number of shares to be sold/bought. For a quote request the last sale price for the given stock should be returned. The investor process outputs messages of the following form (on their own lines, without the quotation marks) to standard output:
 - “No Price” - if there is no available price for a quote request
 - “Quote: \$N.nn” - for a quote request where the price is known (N.nn is replaced by the price).
 - “Sale at \$N.nn” - if a sell or buy is successful (N.nn is replaced by the price).
 - “No Sale” - if the sell or buy is not successful

Other information may be output to the standard error stream, but no other messages are to be output to standard output. If an empty string is entered as the transaction type then the investor process will quit.

The investor interface described above allows the process to be run interactively (by a person) or in an automated manner (as will happen for assessment).

- The share registry process must output (to standard output) a four line summary of each sale transaction. This summary should take the form:
Seller: HIN (Broker)
Buyer: HIN (Broker)
Stock: CODE
Number: nnnnn
where HIN is replaced by the seller/buyer holder identification number, Broker is replaced by the name of the seller/buyer’s broker, CODE is replaced by the stock code, and nnnnn is replaced by the number of shares transferred (no commas or decimal points).
- The broker and stock exchange processes are not expected to output anything but may do so for your debugging purposes (either to standard error or standard output).
- You can assume that investors won’t sell shares that they don’t have and that all data entry happens correctly - i.e. you do not have to check for errors in supplied data or command line arguments. Stock codes are arbitrary - any entered code is assumed to be valid.
- Processes do not need to maintain any persistent state (i.e. they don’t need to remember stock codes, last sale prices etc between invocations).
- Messages are encoded using the XDR format.
- Each process is single-threaded and uses only a single communication end-point (all messages are received on the same port) but servers support multiple clients.

- Your message format designs must minimize the number of wasted bytes (bytes that don't convey meaningful data), i.e. you should not be sending information which is not needed. You may assume that a process that receives a message knows where it comes from and is able to send a reply (i.e. you do not need to encode sender identification information into messages).

Note: it is possible that there are inconsistencies in the above requirements and/or that not all details have been specified. Please ask if you are unsure of the requirements. Please monitor your email, the course newsgroup, or the course website for clarifications and/or corrections to the above information. It will be assumed that students see such email or postings by the end of the next business day. Requirements changes/clarifications emailed and/or posted by one of the teaching staff before 4pm Wednesday September 9 is considered to be part of the assignment requirements.

Part A Tasks

1. You need to **annotate the figure** shown on page 6 to show the communication that occurs between the processes. A directed arc or arrow () should be drawn between processes to indicate message passing of some sort from the process at the origin of the arrow to the process at the head of the arrow. (If communication is bidirectional, a *separate* arrow should be drawn in the other direction also.)
2. You must complete a **table showing the communication primitives** which are used at each end of each arc. Your table should be in the following format and there should be at least one row in the table for every arc/arrow shown in your communication figure. The last column should list the name (or number) of the message format(s) that are used for that communication.

These are the formats to be designed in step 4 below.

Sending Process	Send Primitive	Receiving Process	Receive Primitive	Message Format Names
e.g. Investor
...	Eg. RPC

Remember, possible sending primitives are blocking send, non-blocking send, RPC call, and RPC reply. Possible receiving primitives are blocking receive, non-blocking receive, RPC server accept, and RPC call. (Note that RPC call is both a sending and receiving primitive.)

Marks will be given for the use of the primitives that most closely resemble the communication semantics indicated in the specification. You must pay particular attention to the difference between remote procedure calls and message passing. If communication looks on the client side like an RPC call (RMI), you should use the RPC call primitive - independent of whether the server is an RPC (RMI) server. If a process behaves like an RPC server you should use the RPC server accept and RPC reply primitives, independent of whether the clients use RPC call to communicate with it. (In other words, you may mix and match RPC primitives on one end with message passing primitives on the other if appropriate).

3. You should write a **short explanation** (around a paragraph, at most one page) which justifies your selection of communication primitives.

4. You should design the format of the messages that will be used in the communication between processes. For **each** message format name (or number) you've listed in the table above you should specify the fields that make up the message. The field specification should include the

- name/description of the data (e.g. item ID)
- XDR type of the data (e.g. integer, character, fixed length string, etc)
- size of the field in bytes (or range of sizes if the size is variable)

You should also specify the overall size of the message (or range of overall sizes if the size is variable).

(Don't forget the padding rules of XDR.)

For example, the format of all the Name Server's messages (registration, lookup requests and replies) is partially shown below (field sizes are omitted).

ns_all

operation (character)	name (fixed length string)	IP-address (integer)	port-number (integer)
---------------------------------	--------------------------------------	--------------------------------	---------------------------------

5. You should write a short discussion (around one paragraph, at most one page) describing any **assumptions** that you've made and/or any **limitations of your design**.

Assessment Criteria for Part A

Provided your assignment is submitted following the instructions below, it will be marked according to the following criteria:

- **Identification of communicating processes and directions** (15 marks)

For each of the 5 process types, the correct presence/absence of arrows from and to other processes (3 marks for each process).

- **Choice of communication primitives** (30 marks)

Proportion of primitives correctly selected and justified. Your mark will be calculated based on the number of lines in your primitive specification table, as

$$15 \text{ marks} * (\text{Number of correct receive primitives} + \text{Number of correct send primitives})$$

$$\frac{\text{Max/Number of lines in your primitive specification table } or \\ \text{Number of arrows in your communication figure } or \\ \text{Number of lines in correct primitive specification table}}{\text{Number of lines in your primitive specification table}}$$

- **Design of message formats** (50 marks)

• 30 marks for selection of appropriate data types meeting the given specification (30 marks awarded if there are no missing or incorrect fields and no missing formats. 2 mark deduction for the first mistake; 1 mark deduction for following mistakes. Minimum mark 15 out of 30 if at least 50% of the message formats are completely correct. Minimum mark 5 out of 30 if at least one message format is completely correct.)

• 20 marks for message field sizes and overall message sizes (18 marks awarded if there are no missing or incorrect field/message sizes. 1 mark deduction for each mistake. Minimum 10 out of 20 if at least 50% of the message formats are correctly sized. Minimum mark 2 out of 20 if at least one message format is correctly sized.)

- **Statement of limitations and assumptions** (5 marks)

5 marks awarded if reasonable limitations/assumptions are stated (that aren't a restatement of any of the specifications and don't violate the specifications). 1 mark deduction for each mistake or omission. Minimum 2 marks out of 5 if at least one correct and reasonable limitation/assumption is stated.

Submission Instructions for Part A

Your submission must consist of:

- page 7 of this document for answering task 1 (make this the first page of your submission after the standard ITEE coversheet.)
- the table showing the choice of communication primitives
- an explanation of the choice of communication primitives (at most one page)
- message format design (and sizes)
- a statement of assumptions/limitations (at most one page)

You must follow the following rules. Failure to do so may result in a mark of 0.

- Use only A4 paper
 - **Staple** only in the top left corner, as indicated
 - Do not use binders, folders, clear plastic sheets, clips, etc.
 - Attach a standard ITEE cover sheet. • You **MUST** sign and date the declaration regarding originality.
- Failure to do so will result in a mark of 0 for Part A of this assignment.**

Submission Time and Place

Assignments are due at **4pm on Monday September 14**. You must submit Part A of your assignment to the COMS3200 / COMS7201 submission box on the first floor of the GPSouth building. You are advised to keep a copy of your assignment.

Investor

Broker

Name
Server

Stock
Exchange

Share
Registry

Part B (10%)

The aim of Part B of the assignment is for you to gain experience in network programming using socket level primitives for the TCP protocol. To this end, you are required to implement the networked application described in part A. As the emphasis of this assignment is on communication using TCP sockets, the processing of requests in the servers are drastically simplified compared to real servers. The functionality of the clients and servers is the same as described in part A. There is however a difference in the communication primitives. You will only use TCP sockets, not any other mechanisms (such as RPC Calls or RMI).

Specification

Languages

You must implement the whole application in either Java or C using socket level primitives. All communication is to be TCP based. The servers and clients can **NOT** be multi-threaded (i.e. you cannot use *fork()* for C or threads in the Java implementation or any other systems command which will make the program multi-threaded. You can however use *select()* for C and the equivalent non-blocking IO commands in Java).

Java Requirements

Your implementation must consist of 5 classes: Investor, Broker, Exchange, Registry, and NameServer which will be in files Investor.java, Broker.java, Exchange.java, Registry.java and NameServer.java. Each of these classes will contain a `public static void main(String args[])` method. (There may be additional classes present in these files also, but you may not use additional files.)

C Requirements

If you choose to use C, your implementation must consist of any needed .h and .c files as well as a Makefile. When `make all` is performed, your Makefile must compile and link your source files into five executable programs: Investor, Broker, Exchange, Registry, NameServer. **Your code must be compiled with the `-Wall` and `-ansi` flags.**

Investor

The Investor must satisfy the following requirements:

- It must accept 2 command line arguments:- `<investor name>` , `<Broker Name>`..
`<investor name>` is a unique holder-identification number (HIN) which has 10 characters, while `<Broker Name>` is a string of 20 characters, who will respond to the investor's request for quote, sell and buy.

`Investor <investor name> <Broker Name>`

- If the arguments aren't of the expected number and format (e.g. the two arguments are not strings) then your program should print the following message to **standard error** and exit with an exit status of 1:

`"Invalid command line arguments for Investor\n"`

- The Investor first contacts the Name Server to get the IP address and port number of the specified Broker,. If the Investor cannot contact the Name Server at the specified address and

port number, it should print the following message to **standard error** and exit with an exit status of 1:

```
"Cannot connect to name server located at <Name Server's IP
Address>: <port>\n".
```

- Stocks are identified by a 6 letter code. Share price are dollars and cents and will be in the range of 0.01 to 1,000,000.00.
- An investor process must prompt (via standard error) for user input (via standard input) as follows. It must first prompt for the transaction type - either "q" for quote, "s" for sell, or "b" for buy. It should then prompt for the stock code. If the transaction is a sell or buy transaction, it should then prompt for the number and price of shares to be sold/bought. For a quote request the last sale price for the given stock should be returned. The investor process outputs messages of the following form (on their own lines, without the quotation marks) to standard output:
 - "No Price" - if there is no available price for a quote request
 - "Quote: \$N.nn" - for a quote request where the price is known (N.nn is replaced by the price).
 - "Sale at \$N.nn" - if a sell or buy is successful (N.nn is replaced by the price).
 - "No Sale" - if the sell or buy is not successful
- If the Investor cannot contact the Broker at the given IP address and port number returned by the Name Server, it should print the following message to **standard error** and exit with an exit status of 1:

```
"Cannot connect to the broker located at <Broker's IP
Address>: <port>\n".
```

Name Server:

The Name Server must satisfy the following requirements:

- [It must have no command line argument.](#)

```
NameServer
```

- If the arguments aren't of the expected number, then your program should print the following message to **standard error** and exit with an exit status of 1:

```
"Invalid command line arguments for NameServer\n"
```

- [Use "10000+last 4 digits of your student number" \(or nearby number\) as the port number used by the Name Server](#)
- Your Name Server should listen on this given listening port number for incoming connections from other processes. If your Name Server is unable to listen on the given port number, it should print the following message to **standard error** and exit with an exit status of 1:

```
"Cannot listen on given port number <port>\n" where <port> is replaced
by the listening port number.
```

- The Name Server will print the following message to **standard out** and wait for any incoming connections on the given port <listening port> if it is able to listen on the port.

"Name Server waiting for incoming connections ..\n"

- The Name Server should accept two types of messages lookup queries and register queries (Note: the messages do not have to be named as such).
- Upon receiving a valid register request the Name Server will store the name to address mapping in the format designed in part A. You should assume that each server have a unique name. i.e. there will never be a case where two different servers with the same name are running at one time.
- Upon receiving a valid lookup message the Name Server will search its list of registered servers. If the server cannot be found then the Name Server will reply with an error message:

"Error: Process has not registered with the Name Server\n", which will be sent back to the connecting process.

- The Name Server must be able to handle requests in which the connecting client sends rubbish data, (i.e. not in the form the Name Server was expected) which is done by closing the connection. The Name Server is not expected to exit unless it encounters problems unrelated to communication (e.g. running out of memory). These circumstances will not be tested.

Broker Server:

The Broker Server must satisfy the following requirements:

It must accept 1 command line argument--<broker name>.
<broker name> is a string of 20 characters.

Broker <broker name>

- If the arguments aren't of the expected number and format then your program should print the following message to **standard error** and exit with an exit status of 1:

"Invalid command line arguments for Broker\n"

- Your Broker Server should listen on the listening port number for incoming connections from Investors. If your Broker Server is unable to listen on the given port number, it should print the following message to **standard error** and exit with an exit status of 1:

"Cannot listen on given port number <port>\n" where <port> is replaced by the listening port number.

- The broker needs to register with the Name Server.
- Brokers respond to investor requests. For a quote request, the broker returns the last known sale price for the given stock. If no last sale price is on record for that stock, then an indication of this is returned to the investor. If no stock is specified, the broker returns all of the last sale prices it has on record (possibly 0). For buy and sell requests, the broker passes the request on to

the stock exchange and returns the status of the transaction to the investor when that information is returned by the stock exchange. The broker should deal with other requests (from other investors) if any in the mean time. Brokers maintain a record of last sale prices for all stocks that have been sold (since the process started).

- If a transaction is successful at the Exchange Server, all brokers will be informed of the last sale price of a specified stock so each broker will update the record of last sale prices for all stocks. If a transaction is not matched within the specified time of it arriving at the stock exchange, then the stock exchange notifies the broker who submitted the request that a sale/buy was not able to be completed.

Exchange Server:

The Exchange Server must satisfy the following requirements:

It must have 1 command line argument-<request-hold-time>. <request-hold-time> is the request-hold time in seconds.

Exchange <request-hold-time>

- If the arguments aren't of the expected number and format, then your program should print the following message to **standard error** and exit with an exit status of 1:

"Invalid command line arguments for Exchange\n"

- Your Exchange Server should listen on the listening port number for incoming connections from other processes. If your Exchange Server is unable to listen on the given port number, it should print the following message to **standard error** and exit with an exit status of 1:

"Cannot listen on given port number <port>\n" where <port> is replaced by the listening port number.

- The Exchange Server needs to register with the Name Server. It should use the name "Exchange".
- The Exchange Server receives buy and sell requests from brokers (forwarded from investors) and tries to match such requests (based on the particular stock, the number of shares to be sold/bought and any price information included in the request). Only complete parcels of shares are sold/bought, e.g. a request to buy 200 shares can be met only by selling a package of 200 shares, not by selling two packages each of 100 shares or by splitting up a larger package. The sale price is determined as follows:

Minimum selling price per share (optional)	Maximum buying price per share (optional)	Last sale price	Sale price
Not specified	Not specified	No previous sale	No sale can occur
		Value known	Last sale price
Not specified	Maximum specified	-	Maximum specified buy price
Minimum specified	Not specified	-	Minimum specified sell price
Minimum specified	Maximum specified (less than minimum selling	-	No sale can occur

	price)		
Minimum specified	Maximum specified (greater than or equal to minimum selling price)	-	- Average of minimum sell price and maximum buy price (rounded to the nearest cent)

- The stock exchange will hold a request for up to 20 seconds (or some other time specified at process run time) in an attempt to find a matching request in order to complete the transaction. If a match occurs within this time, the stock exchange will inform the brokers of the selling/buying investors of the sale and of the agreed sale price. All brokers will be informed of the last sale price of that stock. If a transaction is not matched within the specified time of it arriving at the stock exchange, then the stock exchange notifies the broker who submitted the request that a sale was not able to be completed.
- After a sale, the stock exchange will send information to the share Registry Server to inform it about the seller, the buyer, the names of their brokers, the stock code, and the number of transferred shares.

Registry Server:

The Registry Server must satisfy the following requirements:

- It must have no command line argument.

Registry

- If the arguments aren't of the expected number, then your program should print the following message to **standard error** and exit with an exit status of 1:

```
"Invalid command line arguments for Registry\n"
```

- Your Registry Server should listen on the listening port number for incoming connections from other processes. If your Exchange Server is unable to listen on the given port number, it should print the following message to **standard error** and exit with an exit status of 1:

```
"Cannot listen on given port number <port>\n" where <port> is replaced by the listening port number.
```

- The Registry Server will send a register request to the Name Server. It should use the name "Registry".

- The Registry server will receive share registry information from Exchange Server. The information includes the seller, the buyer, the names of the broker, the stock code, and the number of transferred shares. Then registry process must output (to standard output) a four line summary of each sale transaction. This summary should take the form:

```
Seller: HIN (Broker)
Buyer: HIN (Broker)
Stock: CODE
Number: nnnnn
```

where HIN is replaced by the seller/buyer holder identification number, Broker is replaced by the name of the seller/buyer's broker, CODE is replaced by the stock code, and nnnnn is replaced by the number of shares transferred (no commas or decimal points).

Hints

- You may wish to start from the examples server/client available on the COMS3200/7201 website. You are free to use this code however you like. You may also refer to the examples in lecture 10 of the course comp2303 (www.itee.uq.edu.au/~comp2303/lectures).
- Processes can find their IP addresses using the `gethostbyname()` function.
- The port numbers used by Broker, Exchange and Registry should be allocated by the system. Please refer to the `servereph.c` example available on the course website.
- Each Broker can establish a connection with Exchange and keep it open for all of its communication with Exchange. The same approach can be taken for the communication between Exchange and Registry.
- Processes on UNIX - you should be careful not to leave processes running on shared UNIX machines - they may use up resources. You may use the `ps` or `jobs` command to discover details of the processes you own and then use the `kill` command to kill off unwanted processes. Consult the UNIX man pages for details. .
- You may wish to consider using the netcat utility on agave to interact with your client and/or server. Type `man netcat` on agave to see the manual page for netcat.

Assessment Criteria

Provided your assignment is submitted following the instructions below, it will be marked according to the following criteria. You must pay careful attention to the details of any required behaviour. Part marks may be awarded for a given criteria if the specification is partially met. All marking will be performed in a UNIX (Solaris) environment, specifically `agave.students.itee.uq.edu.au` and it is expected that your code will work in this environment. Note that some criteria can only be tested for if other criteria are met (e.g. connections established).

Investor (15 marks)

- Code compiles successfully (1 mark)
- Investor correctly deals with invalid number and format of command line arguments (1 marks)
- Investor correctly prints out error message and exits with the correct status when unable to contact the Name Server (1 marks)
- Investor correctly displays the response for a quote request (4 marks)
- Investor correctly displays the response for a sale request (4 marks)
- Investor correctly displays the response for a buy request (4 marks)

Name Server (15 marks)

- Code compiles successfully (1 mark)
- Name Server correctly deals with invalid number and format of command line arguments (1 mark)
- Name Server correctly deals with being unable to listen on the given port number (1 mark)
- Name Server correctly responds to a valid query (4 marks)
- Name Server correctly registers an item when receiving a valid register request (4 marks)
- Name Server prints out an error when it receives an invalid register or query request (4 marks)

Broker Server (25 marks)

- Code compiles successfully (1 mark)
- Broker correctly deals with invalid number of command line arguments (1 mark)

- Broker correctly deals with being unable to listen on the given port number (1 mark)
- Broker correctly sends a valid register request to the Name Server (2 marks)
- Broker correctly sends requests to the Exchange Server (5 marks)
- Broker correctly responds to a quote request sent from an Investor (4 marks)
- Broker correctly responds to a sale request sent from an Investor (4 marks)
- Broker correctly responds to a buy request sent from an Investor (4 marks)
- Broker updates the record of last sale when receiving it from the Exchange Server (4 marks)

Stock Exchange Server (35 marks)

- Code compiles successfully (1 mark)
- Exchange Server correctly deals with invalid number and format of command line arguments (1 marks)
- Exchange Server correctly deals with being unable to listen on the given port number (1 mark)
- Exchange Server correctly sends a register message to the Name Server (2 marks)
- Exchange Server correctly sends a Registry lookup request to the Name Server (2 marks)
- Exchange Server correctly deals with a buy request forwarded from a broker (12 marks)
- Exchange Server correctly deals with a sale request forwarded from a broker (12 marks)
- Exchange Server sends a valid share registry information to the Registry Server (4 marks)

Share Registry Server (10 marks)

- Code compiles successfully (1 mark)
- Registry Server correctly deals with invalid number and format of command line arguments (1 marks)
- Registry Server correctly deals with being unable to listen on the given port number (1 mark)
- Registry Server correctly sends a valid register message to the Name Server (2 marks)
- Registry Server correctly deals with information received from the Exchange Server (5 mark)

Penalties that may be applied

- Code must be modified by marker to permit compilation and/or marking (-1 to -50 depending on the severity of the change required. Deduction is at the discretion of the course coordinator whose decision is final.)

Submission Instructions for Part B

Assignments are due at 4pm on Monday September 14. You must submit Part B electronically via <http://submit.itee.uq.edu.au>. Only the last of your submissions will be marked. Your submission time will be considered the time of the last of your submissions. Assignments submitted in any other manner (e.g. email) will be discarded.

You are advised to keep a copy of your assignment.

Late Submission of Part A or Part B

Late submission will be penalized by the loss of 10% of your Part A or Part B assignment mark per working day late (or part thereof). In the event of exceptional personal or medical circumstances that prevent on-time hand-in, you should contact the lecturer and be prepared to supply appropriate documentary evidence (e.g. medical certificate). Late Part A submissions must be made directly to the lecturer or tutors and late electronic submissions must be made via <http://submit.itee.uq.edu.au>. Emailed assignments will be discarded.

Modifications to Part A and Part B Requirements

Note: it is possible that there are inconsistencies in the above requirements and/or that not all details have been specified. Please ask if you are unsure of the requirements. Please monitor your email, the course newsgroup, or the course website for clarifications and/or corrections to the above information. It will be assumed that students see such email or postings by the end of the next business day. Requirements changes/clarifications emailed and/or posted by one of the teaching staff before 4pm Wednesday September 9 are considered to be part of the assignment requirements.

Academic Merit, Plagiarism, Collusion and Other Misconduct

You should read and understand the statement on academic merit, plagiarism, collusion and other misconduct contained within the course profile and the document referenced in that course profile. You should note that this is an **individual assignment**. **All submitted source code will be subject to plagiarism and/or collusion detection.** Work without academic merit will be awarded a mark of 0.

Assignment Return: Assignment feedback arrangements will be advised later.