The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2012

COMS3200 / COMS7201 – Assignment One

Due: 2pm Friday April 26, 2013
Assignment weight 17%

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/index.html?page=138114

Part A (5%)

The aim is to design a system of communicating processes using client/server RPC and message passing. You must choose appropriate communication primitives and design suitable message formats. For this part of the assignment you should assume that you have a programming environment in which both message passing and RPC/RMI are available and that blocking and non-blocking primitives exist for sending and receiving. The available communication primitives are assumed to be built on top of a reliable and connection-less 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 six communicating processes in the system (plus the name server). These correspond to

- Travel Agent (manual)
- Travel Agent (automatic)
- Airline QF
- Airline VA
- Bank B
- Reservation Service R

The following requirements must be supported:

- Travel Agents support two customer operations - fare enquiry and fare purchase
- Travel Agents communicate only with the Reservation Service (and the Name Server)
- The Reservation Service does not store flight details or reservations. All requests received from travel agents are passed on to one or both airlines.
- Fare enquiries consist of an origin airport, a destination airport and an optional preferred airline.
- If no preferred airline is specified the reservation service passes the enquiry on to both airlines and returns the results to the requesting travel agent. If a preferred airline is specified, the enquiry is only passed to that airline.
- The result of a fare enquiry is a list (with 0 or more entries) of flight/fare options returned to the requesting travel agent. A flight/fare option consists of a flight number, origin airport, departure time, destination airport, arrival time and a price.
- A fare purchase operation involves the transmission of an airline name, flight number, origin airport, destination airport, passenger name, fare price and credit card number to the airline involved (again, via the reservation service). If a seat is available, the airline will pass the financial details (passenger name, fare price and credit card number) on to the bank B which will return a confirmation number. If a seat is unavailable, the requesting travel agent will be informed.
- Credit card transactions are considered to be always successful. The bank generates a confirmation number, and prints this and the financial details received to the screen. It then sends the confirmation number back to the airline.
- Airports are represented with 3 letter codes (e.g. BNE, SYD, MEL)
- Airlines are represented with 2 letter codes (e.g. QF, VA)
- Flight numbers consist of the airline code plus a 1 to 4 digit number (e.g. QF1, VA7)
- Prices are dollars and cents and will be in the range 0.00 to 5000.00.
- Credit card numbers are 16 digits long
- As it is easier to program blocking receive than non-blocking receive (and very often it also provides a better performance) for each process you should consider whether it would work correctly (i.e. according to the specification) if blocking receive is used.

Implementation Requirements and Assumptions

These requirements and assumptions don’t affect the communication, but do specify how the processes are to operate.

- One travel agent process must prompt for user input as follows. It must first prompt for the transaction type - either “p” for purchase or “e” for fare enquiry. Based on the transaction type entered, it should then prompt for the other required details.
- If a user enters an empty string as transaction type then the interactive travel agent process will quit.
- The other travel agent process must generate and send transactions automatically - using simple example data. You may insert timer delays in the code to slow down the operation if you like.

- The travel agent processes one transaction at a time (it does not start a new transaction until the previous fare enquiry or fare purchase has finished.
- Airline processes start with some hardwired set of flight data. No persistent storage across runs is necessary. Airlines do not need to store passenger booking details - they should just print them out. Airlines should store the flight details and number of available seats. Airlines should not sell more seats on a flight than they have available. It is acceptable for airlines to return flight information in response to a fare enquiry even if no seat is available on that flight.
- You may assume that the user always supplies valid responses to the travel agent - i.e. you don't have to do data entry validation.
- The Travel Agents, Airlines, Reservation Service and Bank have to register their communication endpoints (IP address, port) with the Name Server and the Name Server has to acknowledge the operation. The Name Server keeps a mapping from a server name to its endpoint. The processes (servers or clients) that need to communicate with a given server send a request (Lookup request) to the Name Server requesting a communication endpoint of a server with a given name. The lookup request should be sent before any communication with the server starts and the address should be cached afterwards. Clients do not need to register with the name server. The Name Server's endpoint should be passed as a parameter to the servers that they know how to communicate with it when they need to register.
- All servers have only a single communication end-point (port). This means that if a server can receive two (or more) different types of messages, then it must have a way of differentiating between them (and they can't be received by two different receiving primitives). The exception is the Name Server with which the processes have to communicate to learn about communication end-points. While the same port has to be used for this communication the processes may use different communication primitives for communicating with the name Server than with the remaining processes.
- Note that processes are single-threaded but may need to supports multiple clients communicating with it "simultaneously". For example, the Reservation Service can process a new request from a Travel Agent while it is waiting for information from Airlines about operations which were previously started on behalf of other Travel Agents.
- You must specify any assumptions (additional requirements) that you make. It is permissible to set reasonable limits on variable lengths and variable numbers.
- 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 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 Monday April 22 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

communication 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 used in point 4.**
e.g. Client	e.g. 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, you should use the RPC call primitive - independent of whether the server is an RPC 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. flight_number)
- 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.)

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 (10 marks)
For each of the 5 process types, the correct presence/absence of arrows from and to other processes (2 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

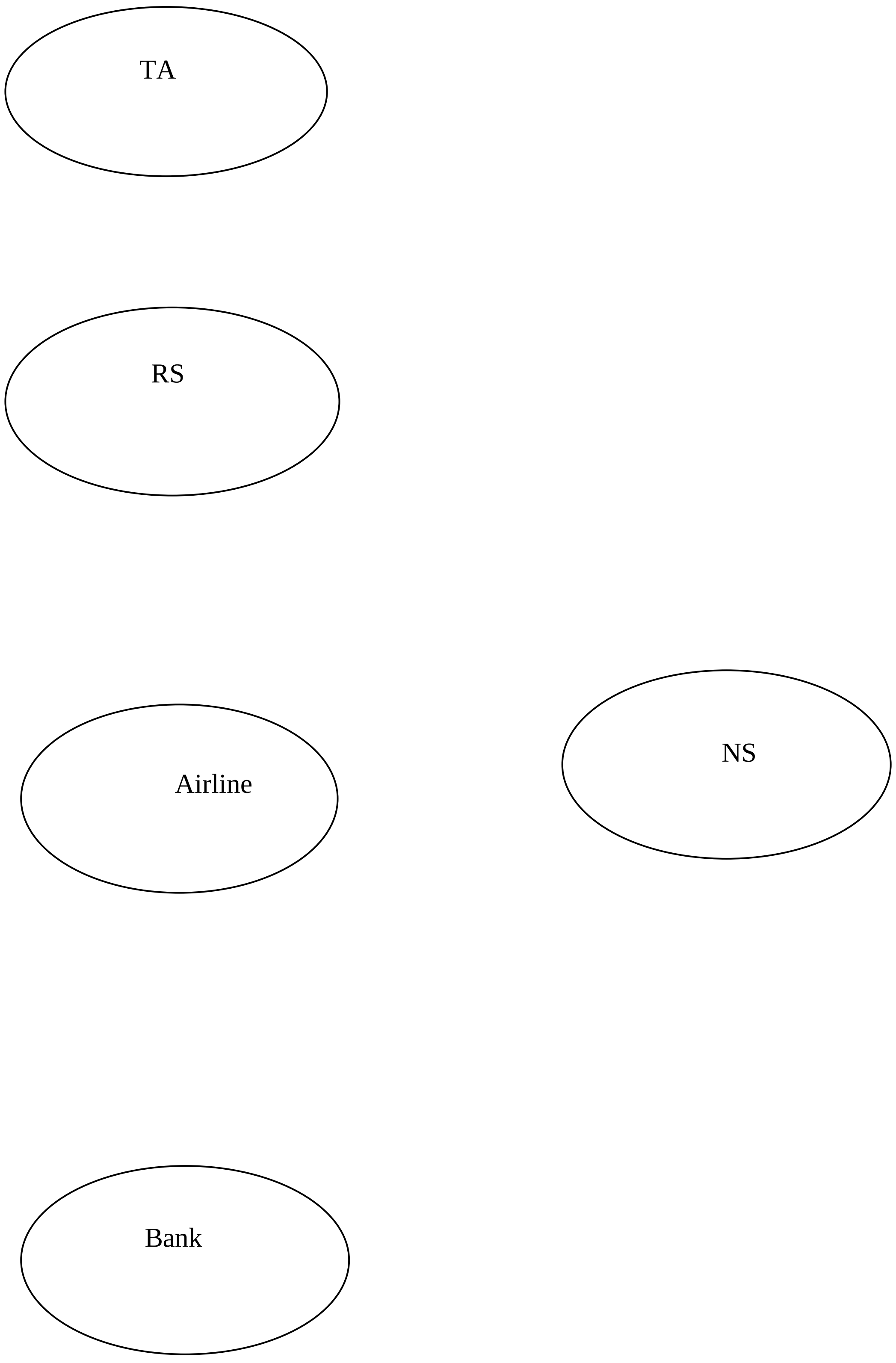
$$\frac{\text{15 marks *(Number of correct receive primitives + Number of correct send primitives)}}{\begin{array}{c}\textit{Max[}\text{Number of lines in your primitive specification table } \textit{or} \\ \text{Number of arrows in your communication figure } \textit{or} \\ \text{Number of lines in correct primitive specification table]}\end{array}}$$

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 (20 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 (10 marks)
10 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 10 if at least one correct and reasonable limitation/assumption is stated.

What to submit for Part A

Your submission must consist of:
• the diagram that shows the processes and their communication
• 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)


TA
RS
Airline
NS
Bank

Part B (12%)

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 processes 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/RMI Calls).

Note: you do not need to use the same communication primitives described in Part A in your implementation (Part B); Part A describes an optimal environment with all primitives available, whereas Part B is the actual implementation in Java.

Specification

You must implement all the components in Java 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 threads in the Java implementation or any other systems command which will make the program mutli-threaded). You can use non-blocking IO commands in J`ava.

Java Requirements
Your implementation must consist of five classes: `TravelAgent, Airline, Bank, ReservationService,` and `NameServer` which will be in the files `TravelAgent.java, Airline.java, Bank.java, ReservationService.java,` and `NameServer.java`. Each of these classes will contain a `public static void main(String args[])` method. You may use additional classes and Java files as needed but at a minimum your solution must include all of these files. Your Java classes should not be part of any package.

Travel Agent
The agent can handle multiple requests per invocation.
The client must satisfy the following requirements:

- It must accept three (3) command line arguments:- the port number on which the name server is listening on for incoming connections, the hostname which the name server is located on, and whether this instance of the TravelAgent class is expecting manual input, or if it should perform transactions automatically. For instance:

```
TravelAgent <NameServer port number> <NameServer hostname>
<Manual or Automatic>
```

- If the arguments aren't of the expected number (3 after the command name) and form (first argument is integer between 1024 to 65535 inclusive, second and third arguments are strings i.e "localhost manual") then your program should print the following message to **standard error** and exit with an exit status of 1:

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

- If your travel agent cannot contact the Name Server at the given hostname 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 <hostname>:<port>\n”`
  where <hostname> is replaced with the Name Server’s hostname and <port> is replace with the given port number for the Name Server.

- Your travel agent must then present the available transaction types to the user through **standard out**; these are ‘purchase’ and ‘enquiry’. The process will then wait for a user string to act on (‘p’, ‘e’, or empty string).

- For an enquiry, the process will prompt the user for an origin airport, a destination airport, and optionally, a preferred airline. The process will then send the request(s) to the reservation service, which will then forward the request(s) to one (or both) airlines. It then waits for a response from the reservation service, and displays this to the user, and will then prompt for a new transaction.

- For a purchase, the process will prompt the user for an airline, flight number, origin airport, destination airport, passenger name, fare price and credit card number. This will then be sent to the reservation service, which will send these details to the appropriate airline. It will then wait for a confirmation or rejection message from the reservation service.

- Upon receiving a confirmation message, the travel agent should print out “`Successful booking of <name> on flight number <flight_number> from <origin_airport> to <destination_airport> on <airline>. Confirmation number: <confirmation number>. Enjoy your flight\n`”

- Upon receiving a rejection message, the travel agent should print out “`Sorry, there are no more seats left on flight <flight_number>\n`”

- After receiving and handling either a confirmation or rejection of flight booking, the travel agent should prompt the user for a new transaction.

- If the instance of TravelAgent is to be automatic, then it should perform at least one of each of these transactions, printing the equivalent user input for the command it is executing, and all messages described above where appropriate, i.e:

  ```
  (p)urchase or (e)nquiry (empty string to exit)?
  e
  Please enter origin airport, destination airport (optional:
  preferred airline):
  BNE SYD QF
  ...
  ```
  Note that this is example text; your implementation may differ in wording/syntax

- Upon receiving any error messages from the servers which the travel agent connects to it must print the error message to **standard error** and then if the agent cannot be continued (i.e. it wasn’t an error message informing the travel agent of the success/failure of an enquiry/purchase) then the travel agent will exit with an exit status of 1.

Name Server:
The Name Server must satisfy the following requirements:

- It must accept one (1) command line arguments:- the port number which the Name Server will listen for incoming connections, i.e.:

  ```
  NameServer <listening port number>
  ```

- If the arguments aren't of the expected number (1 after the command name) and form (argument is an integer from 1024 to 65535 inclusive) then your program should print the following message to **standard error** and exit with an exit status of 1:

  `"Invalid command line argument for Name Server\n"`

- Your Name Server should listen on the given listening port number (on all of the machine's IP addresses) 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 will then listen for incoming connections from other processes. The Name Server will 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 message the Name Server will store the details in some format chosen by you, at least the processes name, the port number where it is listening for incoming connections and the IP Address must be stored. Since there can be two travel agents, and two airlines registered with the name server at any one time, you will need to be able to identify which registration information corresponds to which instance of the class, i.e. the Name Server will need to be able to tell the difference between Airline QF and Airline VA.

- Upon receiving a valid lookup message the Name Server will search its list of running servers and try to find the details. 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 when 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.

Airline
The Airline must satisfy the following requirements:

- It must accept three (3) command line arguments:- the port number on which the name server is listening on for incoming connections, the hostname which the name server is located on, the port number which the Airline will listen for incoming connections, and whether or not this instance of the Airline is for QF or VA, i.e.:

  ```
  Airline <NameServer port number> <NameServer hostname> <QF or
  VA>
  ```

- If the arguments aren't of the expected number (3 after the command name) and form (first argument is an integer from 1024 to 65535 inclusive, second argument is a string, i.e "localhost", third argument is a string of either "QF" or "VA") then your program should print the following message to standard error and exit with an exit status of 1:

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

- Your Airline should listen on a system-assigned port number (on all of the machine's IP addresses) for incoming connections from other processes.

- If your Airline cannot contact the Name Server at the given hostname 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 <hostname>:<port>\n"`
  where <hostname> is replaced with the Name Server's hostname and <port> is replace with the given port number for the Name Server.

- After registering with the Name Server the Airline will print the following message to **standard out** and wait for any incoming connections on the given port <listening port>.

  ```
  "Airline waiting for incoming connections ...\n"
  ```

- When a message is sent to the Airline from the Reservation Service, it will check to ensure the message passed is valid.

  If the message is not valid then the Airline can just close the connection (if one still remains) and ignore the message. If the message is valid however, the Airline must check to see if the airports do exist (and the airline knows about them), and if dealing with a purchase, that there is a seat available.

- When dealing with an enquiry, the Airline must report back all of the flights it knows about, going from the supplied origin airport, to the supplied destination airport.

- When dealing with a purchase, the Airline must first check to see if the requested flight has any spare seats left. If it does not, then the Airline sends back a 'flight full' message, and waits for a new request (purchase or enquiry). If the flight does have at least one spare seat left, then it passes the purchase details to the Bank, and awaits a confirmation number. Once it receives this confirmation number, it passes this along to the Reservation Service to make available to the TravelAgent that requested the flight booking.

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

- The Airline, when launched and after determining if it is QF or VA, should load in a 'hardcoded' set of flight values, so that it has information to work with regarding flights. This can be in any format of your choosing, as it is specific to your implementation.

Bank
The Bank must satisfy the following requirements:

- It must accept two (2) command line arguments:- the port number on which the name server is listening on for incoming connections and the hostname which the name server is located on, i.e.:

  ```
  Bank <NameServer port number> <NameServer hostname>
  ```

- If the arguments aren't of the expected number (2 after the command name) and form (first argument is an integer from 1024 to 65535 inclusive, second argument is a string i.e "localhost") then your program should print the following message to standard error and exit with an exit status of 1:

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

- Your Bank should listen on a system-assigned port (on all of the machine's IP addresses) for incoming connections from other processes.

- If your Bank cannot contact the Name Server at the given hostname 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 <hostname>:<port>\n"`
  where <hostname> is replaced with the Name Server's hostname and <port> is replace with the given port number for the Name Server.

- After registering with the Name Server the Bank will print the following message to **standard out** and wait for any incoming connections on the given port <listening port>.

  ```
  "Bank waiting for incoming connections ...\n"
  ```

- The Bank will only expect one (1) type of message, a purchase message from an Airline. Upon checking that the message contains valid information, the Bank will print this information to its console, and generate a confirmation number which is also printed to the console. This can be a random number, the current system time as a number, a random collection of letters and numbers etc. it's up to you. Once it has created this confirmation number, it should also send this back to the Airline that requested the purchase as the response.

- The Bank must be able to handle when the connecting client sends rubbish data, (i.e. not in the form the Bank was expected) which is done by closing the connection.

Reservation Service
The Reservation Service must satisfy the following requirements:

- It must accept two (2) command line arguments:- the port number on which the name server is listening on for incoming connections and the hostname which the name server is located on, i.e.:

  ```
  ReservationService <NameServer port number> <NameServer hostname>
  ```

- If the arguments aren't of the expected number (2 after the command name) and form (first argument is integer from 1024 to 65535 inclusive, second argument is string i.e "localhost") then your program should print the following message to **standard error** and exit with an exit status of 1:

  ```
  "Invalid command line arguments for Reservation Service\n"
  ```

- Your Reservation Service should listen on a system-assigned port number (on all of the machine's IP addresses) for incoming connections from other processes.

- If your Reservation Service cannot contact the Name Server at the given hostname 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 <hostname>:<port>\n"
  ```
  where <hostname> is replaced with the Name Server's hostname and <port> is replace with the given port number for the Name Server.

- After registering with the Name Server the Reservation Service will print the following message to **standard out** and wait for any incoming connections on the given port <listening port>.

  ```
  "Reservation Service waiting for incoming connections ...\n"
  ```

- The Reservation Service must accept connections from many Travel Agents and send and receive messages from other servers. If the messages received do not match any messages which should be sent to the Reservation Service, the Reservation Service can close the connection. The Reservation Service is not expected to exit unless it encounters problems unrelated to communication (e.g. running out of memory). These circumstances will not be tested.

  The Reservation Service must not wait for reply from servers and ignore other message from the other servers or clients, i.e. the Reservation Service will send the message and then go back to waiting for a message on one of the open connections. It should not for example send a message to an Airline and then block waiting for a reply from that Airline.

- The Travel Agent will, at any point in time, send an enquiry type message (containing origin and destination airport), where the Reservation Service will forward the Airline(s). When it receives information from the Airline(s), it will reply with the list of flights from the origin to the destination. The returned results must include the flight number, origin airport, departure time, destination airport, arrival time and a price.

- The Travel Agent will also, at any point in time, send a purchase type message (containing the airline name, flight number, origin airport, destination airport, passenger name, fare price and credit card number), which will also be forwarded to the Airline indicated in the purchase. Any confirmation/rejection messages it receives from this will be returned to the Travel Agent that initiated the request.

Hints

- You may wish to start from the example server/client available on the COMS3200/7201 website (itee.uq.edu.au/~coms3200 - the page is also linked from Blackboard). You are free to use this code however you like.
- You may wish to consider using the netcat utility on moss to interact with your client and/or server. Type man netcat on moss 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 environment, specifically moss.labs.eait.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). You will need to demonstrate Part B if requested by the tutor.

Travel Agent (20 marks)

- Code compiles successfully (1 mark)
- Server correctly deals with invalid number of command line arguments (1 marks)
- TA 1 correctly deals with all types of inputs (2 marks)
- TA 2 generates requests automatically (2 marks)
- TA correctly prints out error message and exits with the correct status when unable to contact Name Server (2 marks)
- TA correctly sends enquiry requests to the Reservation Service (5 marks)
- TA correctly sends purchase requests to the Reservation Service (5 marks)
- TA correctly looks up required Server information from the Name Server (2 marks)

Name Server (15 marks)

- Code compiles successfully (1 mark)
- Server correctly deals with invalid number of command line arguments (1 marks)
- Server correctly deals with being unable to listen on given port number (1 mark)
- Server correctly prints message expected when listening (1 mark)
- Server correctly handles invalid messages (2 mark)
- Server correctly handles valid messages (2 mark)
- Server correctly sends error messages when process is not registered (2 marks)
- Server correctly stores valid information without any error (3 marks)
- Server doesn’t crash for communication cases (2 marks)

Bank (15 marks)

- Code compiles successfully (1 mark)
- Server correctly deals with invalid number of command line arguments (1 marks)
- Server correctly prints message expected when listening (1 mark)

- Server correctly prints out error message and exits with the correct status when unable to contact Name Server (2 marks)
- Server correctly handles invalid messages (1 mark)
- Server correctly handles valid messages (3 mark)
- Server correctly prints transactions (4 marks)
- Server doesn't crash for communication cases (2 marks)

Airline (20 marks)
- Code compiles successfully (1 mark)
- Server correctly deals with invalid number of command line arguments (1 marks)
- Server correctly prints message expected when listening (1 mark)
- Server correctly prints out error message and exits with the correct status when unable to contact Name Server (2 marks)
- Server correctly handles invalid messages (2 marks)
- Server correctly handles valid messages (4 marks)
- Server prints booked flights (3 marks)
- Server correctly communicates with the Bank (2 marks)
- Server correctly communicates with the Reservation Service (2 marks)
- Server doesn't crash for communication cases (2 marks)

Reservation Service (30 marks)
- Code compiles successfully (2 mark)
- Server correctly deals with invalid number of command line arguments (1 marks)
- Server correctly prints message expected when listening (1 mark)
- Server correctly prints out error message and exits with the correct status when unable to contact Name Server (2 marks)
- Server correctly handles enquiry message (6 marks)
- Server correctly handles reservation message (6 marks)
- Server correctly forwards purchase requests to Airlines (2 marks)
- Server correctly forwards enquiry requests to Airlines (2 marks)
- Server doesn't block waiting for a reply from a message it sent when other connections need to be handled (4 marks)
- Server correctly retrieves other Airline servers information from the Name Server (2 marks)
- Server doesn't crash for communication cases (2 marks)

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
The assignment (Part A and Part B) is due at 2pm on Friday April 26 and should be submitted as a zipped file through Blackboard. You are advised to keep a copy of your assignment.

Late Submission
Late submission will be penalized by the loss of 10% of your 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 submissions must be made directly to the lecturer or tutor (Blackboard submission will be unsuccessful).
.

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 unsure of the requirements. Please monitor your email, the course newsgroup, or Blackboard 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 Monday April 22 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 School website. 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.