The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2014

COMS3200 / COMS7201 – Assignment 1
Due: 2 pm, Tuesday 29 April, 2014
Assignment weight 19%

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 – do not 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 (6%)

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

The scenario is a Library Management System (LMS) that supports the library on-line catalog, plus loans of books by registered library users.

There are seven types of processes in the LMS system. They are all single-threaded and have one communication port. The processes are:

- Loans Server (there is one process of this type)
- User Server (there is one process of this type)
- Catalog Server (there is one process of this type)
- Checkin Client (there may be many processes of this type
- Checkout Client (there may be many processes of this type)
- User Registration Client (there may be many processes of this type)
- User Query Client (there may be many processes of this type)

The following requirements are to be supported:

- Each book in the collection is identified by a unique 12 digit item ID (which is also recorded on a barcode on the book).
- Each registered user (or borrower) in the system is identified by a 12 digit user ID (which is recorded on the user's ID card).
- The Catalog Server records the following information for each book in the collection:
    - Item ID (always) - unique 12 digit number (as described above)
    - Authors (if applicable - not all items have an author) - variable length string for each author, (variable number of authors)
    - Title (always) - variable length string
    - Additional publication information (Publisher, year, edition, etc) – stored as one variable length string
    - Catalog Code (used for where the book is placed on the shelves) – stored as a variable length string
- The User Server records the following information for each registered borrower in the library:
    - user ID (always) - unique 12 digit number (as described above)
    - Name - variable length string
    - Address - variable length string
    - Phone Numbers (home, work, mobile) stored as three variable length strings
    - Email address – stored as a variable length string
- The Loans Server records the following information for each book currently on loan or on hold:
    - Item ID of the book
    - User ID of the associated borrower
    - A flag indicating whether the record is a loan or a renewed loan or a hold
    - The due date for the loan, or last date for the hold

- A Registration Client allows a new book to be added to the system by sending updated information to the Catalog Server, and also allows a new borrower to be added to the User Server. All necessary information, including ID number, is entered through the registration client. Updates to information are done by sending a registration with an existing Item ID or User ID.

- A Checkin Client receives a returned book, and updates loan information based the Item ID of the returned book. If a hold is on the book, the operator is notified so that the book can be put aside.

- A Checkout Client deals with a book being borrowed. First the User ID is entered. If there are any overdue books from that borrower, no loans can be made. Then book IDs are read from barcodes, and the books are loaned for a period of four weeks. An exception is if the book is on hold to a different user, then that book cannot be borrowed, and the operator is notified so the book can be put aside. Borrowers are restricted to 20 loans at one time. At the end of a checkout session, a docket is provided with the user's name, the names and authors of all the books borrowed, and the due dates, and any holds. **Note that in this architecture the Checkout Client does NOT communicate directly with the User Server or the Catalog Server.**

- A Query Client allows borrowers to examine the contents of the catalog. The user can search for items in the catalog by
    - Title search using a variable length search string
    - Author search using a variable length search string (single author per query)
    - Author and Title search using a variable length search string (one for the author and one for the title).
    - Catalog Code search using a variable length string
    - All matching entries (up to maximum of the first 20 matches) are returned from the catalog.
- A Query Client also allows users to look at the loan status of a particular book, by entering the Item ID. The book is shown as available, on loan (with due date), or on hold.
- A Query Client allows a user to enter their ID and get a list of their current borrowings and holds.
- A Query Client allows a user to enter their ID, plus the ID of a book in order to put that book on hold for one week from when it is next available. A book can only have one hold at one time.
- A Query Client allows a user to enter their ID, plus the ID of a book in order to renew the loan for a period of four weeks. Books on hold, and books with loans already renewed once cannot be renewed.
- All processes have only a single communication end-point (port). This means that if a process can receive two (or more) different types of messages at some point in time, then it must have a way of differentiating between them (and they can't be received by two different receiving primitives).
- Note that the Loan Server is single-threaded but supports multiple clients communicating with it "simultaneously". For example, the loan server can process a request for a new loan from one checkout client while it is waiting for information from the Catalog Server about a book title in order to satisfy a different checkout client.
- You must specify any assumptions (additional requirements) that you make. It is permissible to set reasonable limits on variable lengths and variable numbers.
- Message formats should be represented in XDR and the message designs must minimize the number of wasted bytes (i.e. bytes that don't convey meaningful data). For example, if some catalog entry fields aren't applicable for a particular item then they shouldn't be sent (although the receiver will need some way of telling which fields are to be expected).

- 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).
- You may assume that processes know the contact details of other processes.

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 (uq.itee.coms3200), and Blackboard announcements 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 2pm Tuesday April 22 is considered to be part of the assignment requirements.

Part A Tasks

1. The first part of this assignment involves **annotating the figure** shown on page 5 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. Checkin Client	…	…	…	…
…	Eg. RPC Call	…	…	…

Remember, possible sending primitives are blocking send, non-blocking send, RPC call, and RPC reply. Possible receiving primtives 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 to a client 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. license 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.)

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

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

• **Identification of communicating processes and directions** (14 marks)
For each of the 7 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

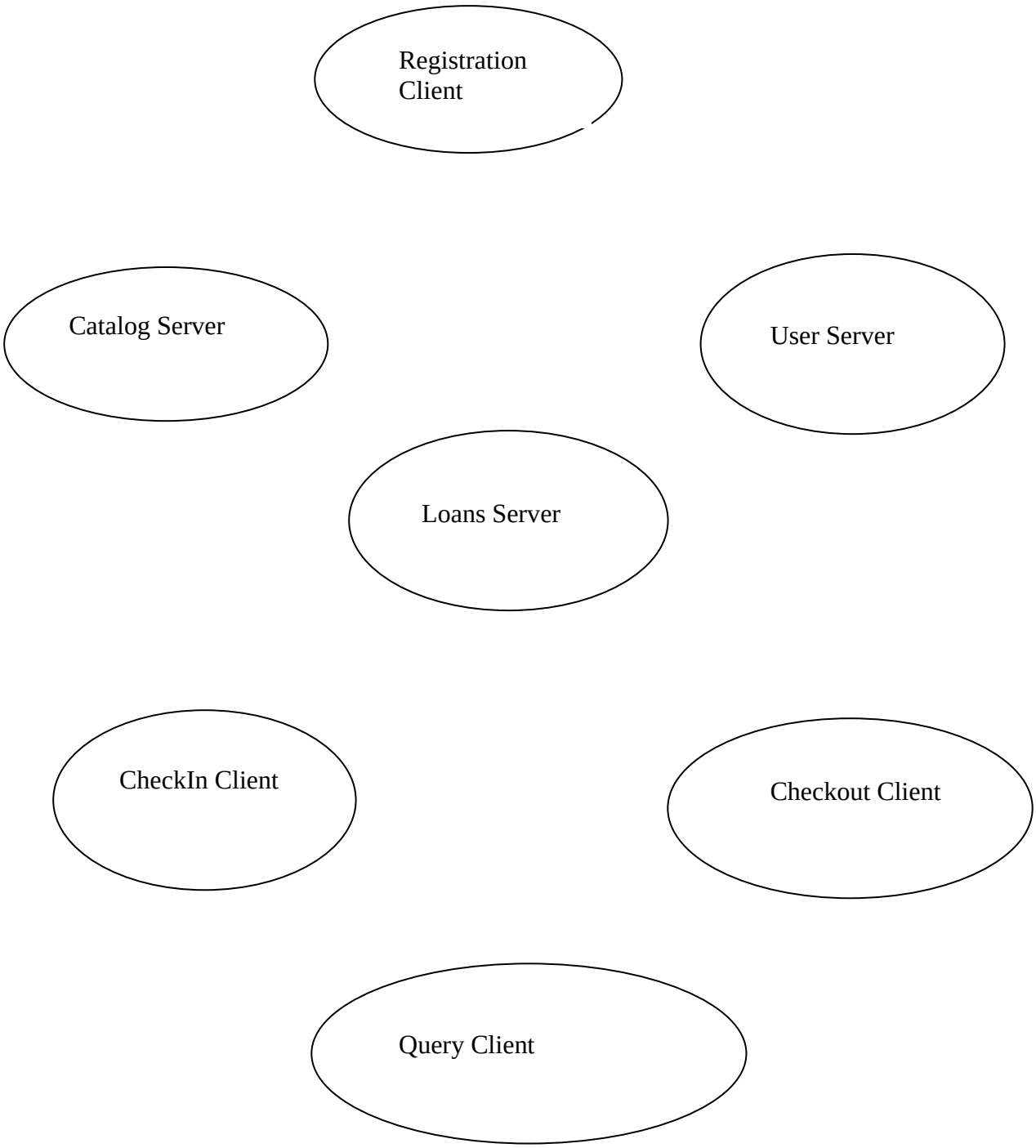
$$\frac{15 \text{ marks} * (\text{Number of correct receive primitives} + \text{Number of correct send primitives})}{Max[\text{Number of lines in your primitive specification table } or \text{ Number of arrows in your communication figure } or \text{ Number of lines in correct primitive specification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

• **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 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** (6 marks)
6 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 1 marks out of 6 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)


Registration Client
Catalog Server
User Server
Loans Server
CheckIn Client
Checkout Client
Query Client

Part B (13%)

The aim of this part 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 Name Server, two clients (QueryClient) and two servers (LoansServer, CatalogServer) in a library loans system. The QueryClient send requests to the LoansServer and the CatalogServer. As the emphasis of this assignment is on communication using TCP sockets, the processing of requests in the servers is drastically simplified compared to real servers. Also note that the functionality of the client and servers is not the same as the servers and clients in Part A. You will only use TCP sockets, not any other mechanisms (such as RPC Calls).

Specification

You must implement all the components in Java using socket level primitives. All communication is to be TCP based. The servers and clients **CANNOT** be multi-threaded (i.e. you cannot use threads in the Java implementation or any other systems commands which will make the program multi-threaded). You can use non-blocking IO commands in Java if needed.

Java Requirements
Your implementation must consist of four classes: `LoansServer`, `CatalogServer`, `NameServer` and `QueryClient` which will be in the files `LoansServer.java`, `CatalogServer.java`, `NameServer.java` and `QueryClient.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.

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

- It must accept one (1) command line arguments:- the port number on 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 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"`

- It should be assumed that the servers and clients know the Name Server's IP address and port and should be able to communicate with it (its port number is passed as a parameter to these processes).

- 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 : the server's name, the port number on which server is listening for incoming connections, and the IP address.

- Upon receiving a valid lookup message the Name Server will search its list of running 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 a situation when the connecting client sends rubbish data, (i.e. data not in the form that the Name Server has 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.

Server - CatalogServer
Your CatalogServer server must satisfy the following requirements:

- It must accept a command line argument:- the port number on which the Name Server is listening on for incoming connections:
  `CatalogServer <NameServer port number>`

- If the argument isn't of the expected number (1 after the command name) and form (integer between 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 arguments for CatalogServer\n"`

- When your CatalogServer starts up it reads the `Catalog-file` file into an internal data structure. It is a very simplified description of Catalog. It describes 10 current books in the system. In this assignment, this file is never updated. Each of the 10 items is described by:
  - `book-id` (integer value)
  - `Title` (string enclosed in double quotes, eg. "A Tale of Two Cities")
  - `Author(s)` (one string enclosed in double quotes, eg. "Mary Smith, Bob Jones")

- CatalogServer should register with the NameServer. If CatalogServer cannot contact the Name Server on the given 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 <port>\n"` where <port> is replaced with the given port number for the Name Server.

- After reading `Catalog-file` and registering with the Name server, the CatalogServer server must listen for incoming requests on its port. If it is unable to do so (e.g., the port is in use by another process or the port number is invalid), then the process must print the message to standard error and then exit:

  `CatalogServer unable to listen on given port`

  If the CatalogServer server is able to start listening for incoming connections , then your CatalogServer should print the following message to standard error:

  `CatalogServer waiting for incoming connections`

- The QueryClient client may then connect to the CatlogServer server. If the CatalogServer accepts the connection request from the QueryClient client, the QueryClient client then sends one of two requests:
  - a search by book-id returns the matching record
  - a keyword search returns all records with the keyword in the title or authors

  You may assume that if a connection is accepted that the request will be received (i.e. you don't have to handle the situation where the request is not received). You should note that this TCP connection should be kept open until CatalogServer sends a reply back. The server closes the connection after sending the reply. The CatalogServer processes the QueryClient commands and returns the requested records.

Server - LoansServer

Your LoansServer server must satisfy the following requirements:

- It must accept a command line argument:- the port number on which the Name Server is listening on for incoming connections:
  `LoansServer <NameServer port number>`

- If the argument isn't of the expected number and form (integer between 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 arguments for LoanServer\n"`

- When your LoansServer starts up it reads the `loans-file` file into an internal data structure. It is a very simplified description of loans. It describes 5 current loans in the system. In this assignment, this file is never updated. A copy of the file is provided on the assignment website. Each of the 5 items is described by:
  - `user-id` (integer value),
  - `book-id` (integer value),
  - `due-date` (integer value interpreted as DDMMYYYY)

-

- Your LoansServer should register with the NameServer. If the LoansServer cannot contact the Name Server on the given 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 <port>\n"` where <port> is replaced with the given port number for the Name Server.

- After reading `loans-file` and registering with the Name Server, the LoansServer server must listen for incoming requests. If it is unable to do so (e.g. the port is in use by another process or the port number is invalid), then the process must print the message to standard error and then exit:

  `LoansServer unable to listen on given port`

  If the LoansServer server is able to start listening for the connection requests, then your LoansServer should print the following message to standard error:

  `LoansServer waiting for incoming connections`

- The QueryClient client may then connect to the LoansServer server. If the LoansServer accepts the connection request from the QueryClient client, the QueryClient client then sends one of two requests:
  - a request for the loans held by a specific user-id.
  - a request for due-date of a specific book-id.

  You may assume that if a connection is accepted that the request will be received (i.e. you don't have to handle the situation where the request is not received). The LoansServer processes the QueryClient request, returns the requested information and closes the connection.

- The QueryClient client may, as a result of the reply, connect to the CatalogServer server, and request appropriate book title and author information for any titles returned from the LoansServer's reply using the same commands described earlier.

Client - QueryClient

Your client (QueryClient) must satisfy the following requirements:

- It must accept three (3) command line arguments:
  - `NameServer-port`
  - `Request-type`
  - `Request-keyword`

- The command line arguments are interpreted as follows:
  - `NameServerport` is the port number on which the NameServer is waiting for connection requests.)

  - `request` is a single letter with the following meaning:
    - L means query all loans from user-ID = `Request-keyword`
    - D means request due-date for book-ID = `Request-keyword`
    - C means request catalog info for book-ID = `Request-keyword`
    - K means keyword search with keyword= `Request-keyword`

- `Request-keyword` is an integer or string (assume no spaces or quotes) as above. Searches can be exact matches, case-insensitivity not required.

- If an incorrect number or type of arguments is given, your process must print the following message to standard error and exit:

```
Invalid command line arguments
```

- The QueryClient should lookup the LoansServer and CatalogServer in the NameServer. If the QueryClient cannot contact the Name Server on the given 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 <port>\n"` where <port> is replaced with the given port number for the Name Server.

Note: you should start the servers first before you start the clients that they are registered by the time the clients are asking for their IP addresses and ports.

- After the NameServer lookup the QueryClient client should try to connect to the LoansServer and/or the CatalogServer. If it is unable to connect (e.g. there is no process listening on that port) then the QueryClient client should print the following to standard error and exit:

```
QueryClient unable to connect to LoansServer
```

Or

```
QueryClient unable to connect to CatalogServer
```

- If the QueryClient client is able to connect to the server(s) it should send request to the appropriate server. This request is determined by the `request` command line argument.

- The QueryClient client should then output the response to the query to standard output, All responses regarding books to the user should include the book-ID, book title and author. Servers should acknowledge connection closing with messages on standard error:

```
LoansServer connection closed
CatalogServer connection closed
```

- Servers should then listen for requests from other clients and reprint messages on standard error:

```
LoansServer waiting for incoming connections
CatalogServer waiting for incoming connections
```

Hint

- You may wish to start from the client/server example available on Blackboard. You are free to use this code however you like.

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.)

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. The application will be tested against test cases provided with the assignment (loans and catalog files). 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.

QueryClient (22 marks)

- Code compiles successfully (2 marks)
- Coding style (neat code layout and code is commented) (2 marks)
- Client correctly deals with invalid command line arguments (1 marks)
- QC correctly prints out error message and exits with the correct status when unable to contact Name Server (2 marks)
- QC correctly looks up required Server information from the Name Server (2 marks)
- There are two (or more) QCs running in the application (4 marks)
- QC correctly sends requests to the LoanServer (3 marks)
- QC correctly sends requests to the CatalogServer (3 marks)
- QC correctly prints responses (3 marks)

Name Server (18 marks)

- Code compiles successfully (2 marks)
- Coding style (neat code layout and code is commented) (2 marks)
- Server correctly deals with invalid 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 sends error messages when process is not registered (2 marks)
- Server correctly stores valid information without any error (2 marks)
- Server correctly responds to registration and lookup requests (3 marks)
- Server doesn't crash for communication cases (2 marks)

LoansServer (30 marks)

- Code compiles successfully (2 marks)
- Coding style (neat code layout and code is commented) (2 marks)
- Server correctly deals with invalid 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 prints out error message and exits with the correct status when unable to contact Name Server (2 marks)
- Server correctly handles valid requests (10 mark)

- Server correctly prints messages to standard error (7 marks)
- Server doesn't crash for communication cases (2 marks)

CatalogServer (30 marks)

- Code compiles successfully (2 marks)
- Coding style (neat code layout and code is commented) (2 marks)
- Server correctly deals with invalid 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 prints out error message and exits with the correct status when unable to contact Name Server (2 marks)
- Server correctly handles valid requests (10 marks)
- Server correctly prints messages to standard error (7 marks)
- Server doesn't crash for communication cases (2 marks)

Submission Instructions

The assignment (Part A and Part B) is due at 2pm on Tuesday April 29 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 submitted by email to the lecturer or tutor (Blackboard submission will be unsuccessful, i.e., will not be recorded properly).

.

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 (uq.itee.coms3200), 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 2pm Tuesday 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.