The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2015

COMS3200 / COMS7201 – Assignment 1
Due: 2 pm, Tuesday 28 April, 2015
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/itee-student-misconduct-including-plagiarism.

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

Part A of the assignment assumes that the programming environment has the above wide scope of semantics of communication primitives (networking APIs). However in a real programming environment you should check what kind of semantics of communication primitives is available and then select the ones that are suitable for a particular application. For example, for message passing in the Java environment there is java.io with blocking semantics and java.nio with non-blocking semantics. In the Microsoft .NET environment there is message passing but also RMI. In the prerequisite course (CSSE2310) you already learned about programming with threads using blocking semantics of communication primitives. This assignment addresses other ways of communicating between distributed processes.

Requirements

The scenario is a Digital Rights Management (DRM) system that supports trading of protected digital media content.

There are six *types* of processes *in the DRM system*. These are:

- Client (there may be many processes of this type)
- Trusted VM Client (there may be many processes of this type)
- Store (there is one process of this type)
- Content Server (there is one process of this type)
- Licence Server (there is one process of this type)
- Tracker (there is one process of this type)

In addition to these processes, there is one process external to the DRM system (for payment services):

- Financial Server (there is one process of this type)

The following requirements are to be supported:

- Each digital media item in the collection is identified by a unique 10 digit item ID
- Each licence to use a media item is identified by a unique 10 digit licence ID
- Clients allow users to both query the catalog of the Store and to buy a digital media item (and its licence). Each Client can only engage in a single catalog query or single purchase at a time - no further queries or purchase requests will be sent to the Store until a reply is received from any previous query or purchase request.
- Queries to the Store take one of the following forms:
    - Keyword search using a variable length search string
    - 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).
    - Item search using an item ID

- The catalog of the Store contains records of all items in the collection. The information stored for each item consists of:
    - Item ID (always) - unique 10 digit number (as described above)
    - Item type (always) - an indication of the type of the item, which might be a song, video, CD etc. (The number of possible item types is less than 256.)
    - Authors (if applicable - not all items have an author) - variable length string for each author (variable number of authors)
    - Title (always) - variable length string
    - Keywords (always) - variable length string
    - Price (always) - floating point double precision
- In response to a query, the Store returns the complete records for the items which match the query.
- A purchase request from a Client to the Store specifies the unique item ID of the media item to be purchased and the credit card details (string of 16 characters). If the purchase is successful, the Store sends back to the Client the protected media item that includes sealed use conditions. The returned media is of the variable-length opaque data type. The sealed use conditions are embedded in this opaque data.
- When the Store receives a purchase request from the Client, it sends the price of the digital media item and credit card details to the Financial Server (a generic transaction interface for a bank). The Financial Server checks that the credit card is valid and sends back the confirmation or rejection of the transaction (string of 30 characters in each case).
- If the transaction is confirmed by the Financial Server, the Store requests the media item (using item ID) from the Content Server - the media item received from the Content Server is sent to the Client. If the transaction is not confirmed then the Store refuses the purchase request from the Client (string of 30 characters).
- A purchased digital media item is played/accessed by the Trusted VM Client. This Trusted Virtual Machine (VM) may take the form of a media player such as Microsoft's Windows Media Player. The Trusted VM Client and the Client are collocated in one node but they do not communicate. When the media item is accessed for the first time, the Trusted VM Client recovers the sealed use conditions (variable length string) from the media item and sends it to the License Server for a unique license to use the content. The License Server returns the license ID, the access rights for the media (variable length string) and expiry date (8 digits).
- The Trusted VM Client sends update messages to the Tracker every time the user uses the protect digital media. It sends the item ID and the date of use (8 digits). This Tracker might represent some industry body that monitors usage to determine popularity of particular media (to, for example, compile a top 100 list).
- 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).
- The Store is single-threaded but supports multiple clients communicating with it "simultaneously". For example, the server may be dealing with a purchase transaction of one Client (which requires

confirmation of the transaction by the Financial Server and delivery of the media item by the Content Server) when a catalog query from another Client arrives. You should assume that the operating system will buffer messages on the port if they are unable to be read by a process immediately. The next available message will be returned when a process next uses a receiving primitive.

- 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 format 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).
- For part A you may assume that processes know the contact details of other processes.

Part A Tasks

1. You need to **annotate the figure** shown on page 7 to show the communication that occurs between the processes. A directed arc or arrow (--->) should be drawn between processes to indicate message sending 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. (There may be more than one row per arc/arrow if different primitives are used between processes at different points in time.) 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. Client	e.g. 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 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 for Part A

Provided your assignment is submitted following the submission instructions, 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.)

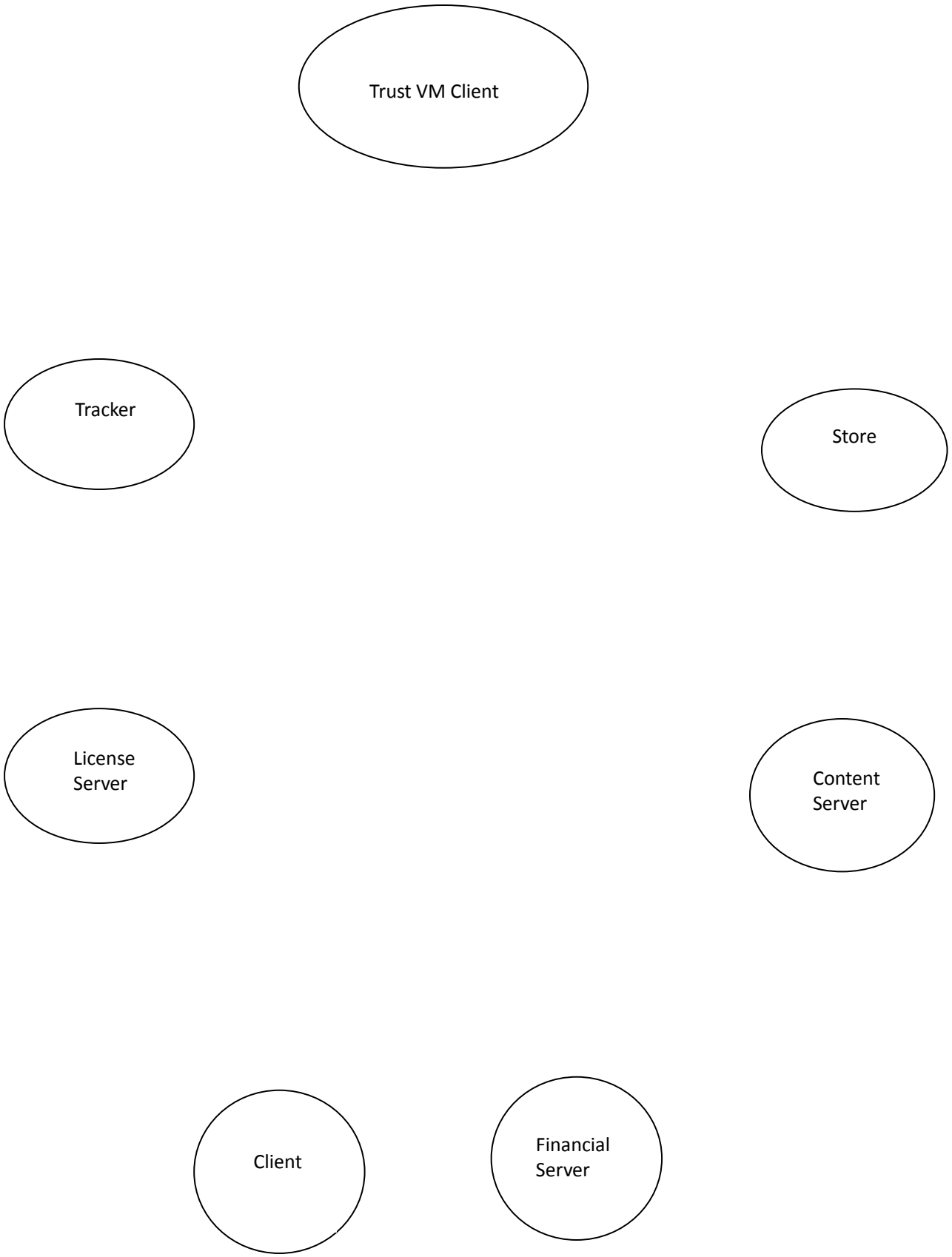
- **Statement of limitations and assumptions (6 marks)**

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)


Trust VM Client
Tracker
Store
License Server
Content Server
Client
Financial Server

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 a subset of the networked application described in Part A plus a Name Server that maps names of Servers into their current IP addresses and ports. The subset of the application which you need to implement comprises one client (Client) and three servers (Store, Bank, and Content). Client sends a request to the Store server while the Store server requests the Bank server (financial Server) to approve transactions and requests the content from the Content server and sends it back to Client. 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 communication 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 5 classes: Client, Store, Bank, Content and NameServer which will be in files Client.java, Store.java, Bank.java, Content.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.)

Name Server:

The Name Server must satisfy the following requirements:

- It must accept one (1) command line argument:
  - NameServer-port
- The command line arguments are interpreted as follows:
  - NameServer-port is the port number which your server will listen.

- 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"

- 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:

  "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 error** 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 (Name, IP and Port). You should assume that each server has 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.

Store

Your Store server must satisfy the following requirements:

- It must accept one command line arguments:
  - Stock-port
  - Stockfile-name
- The command line arguments are interpreted as follows:
  - Stock-port is the port number which your server will listen,
  - Stockfile-name is the file describing the items which can be purchased by Client. The detail content of the file is illustrated afterwards.

- If more than one arguments are given, your process must print the following message to standard error and then exit:

  Invalid command line arguments for Store\n

  Potential problems include: stock-file does not exist. The port number which the Store, Bank and Content Servers will listen on is allocated by the system

- The Store Server needs to register with the Name Server. It should use the name "Store" with the port and IP. If the registration fails, your process must print the following message to standard error and then exit.

  Registration with NameServer failed\n

  Store must also send lookup requests to the Name Server to get the IP address and Port of the server with which it is going to connect (i.e. Bank and Content). If it receives an error message from

NameServer ("process not registered"), it must print the following message to standard error and then exit.

Bank has nor registered\n
or
Content has not registered\n

Store should close its connection with NameServer after registering and getting addresses of Bank and Content.

- Store reads the stock-file file into an internal data structure. It is a very simplified description of stock. It describes 10 items which can be purchased by the Client. Each of the 10 items is described by:
  - item-id (integer value),
  - item-price (floating point single precision),

- Store needs to establish a connection with the Bank server and another connection with the Content server. Each connection should be kept open during the processes' lifetime.

  If it cannot establish connection with Bank or Content, it must print the following message to standard error and then exit.

  Unable to connect with Bank\n
  or
  Unable to connect with Content\n

- After reading the stock-file file and opening the connections to Bank and Content, the Store server must start listening for incoming requests. If it is unable to do so, then the process must print the message to standard error and then exit.:

  Store unable to listen on given port\n

  If the Store server is able to start listening, then your Store should print the following message to standard error:

  Store waiting for incoming connections\n

- Client may then connect to Store. If Store accepts the connection request from Client, Client sends one of two requests:
  - a request for the item-id and item-price of the 10 items in the stock list file
  - a request to buy one item from the stock-list file (credit card number, and item-id are included in this message)

  You may assume that if a connection is accepted then 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 a reply is sent back (or your process exits[1]). You may assume that Client won't close the connection before the reply is sent.

- Store processes Client's commands as follows:

- If the request is for a listing of the items in stock-list, this information is sent back to Client (i.e., item-id and item-price are sent for each item in stock-file)

- If the request is to buy an item then Store sends a request to Bank that has three fields: item-id, item-price and credit-card-number. Bank should then send back a response to Store. This tells the transaction is OK or fails.

- If Bank response is OK, then Store should send a request to the Content server and get the content of the item. The request should include item-id and the response message should include item-id, and item-content. If Store receives a correct response from Content, Store sends the content to Client and closes the TCP connection with Client. If it does not receive a correct response from Content (connection fails), it should send a message to Client containing item-id and the string "transaction aborted". Store should close the TCP connection with Client.

- If Bank response fails, Store should send a message to Client containing item-id and the string "transaction aborted". The server should close the TCP connection with Client.

Your servers including Bank, NameServer, Store and Content are expected to handle multiple requests at a time.

Store must not write any text to standard error (stderr or System.err) except for the messages specified above.

Store may write anything you like to standard output (stdout or System.out). This will be ignored during the marking process.

Bank

Your Bank server must satisfy the following requirements:

- It must accept 1 command line argument:
  - Bank-port
- The command line arguments are interpreted as follows:
  - Bank-port is the port number which your server will listen.
- If an incorrect number of arguments is given, your process must print the message to standard error and exit:

  Invalid command line arguments for Bank\n

- When your Bank starts up it must start listening. 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 following message to standard error and exit:

  Bank unable to listen on given port\n

  If Bank is able to start listening on that port, then it should print the following message to standard error:

  Bank waiting for incoming connections\n

- The Bank server needs to register with the Name Server. It should use the name "Bank" with the port and IP. If the registration fails, your process must print the following message to standard error and then exit.

  Bank registration to NameServer failed\n

- When a request is received, Bank makes an unlawful decision:
  - If item-id is odd, it accepts the transaction and sends confirmation to Store (i.e., sends a single character '1'). Bank also prints item-id and "OK" to standard output.
  - If item-id is even, it rejects the transaction (i.e., Bank sends the single character '0' to Store). Bank also prints item-id and "NOT OK" to standard output.

- Bank is not expected to exit except under the error conditions specified above. Your Bank must not write any text to standard error (stderr or System.err) except for the three messages specified above. Your Bank may write anything you like to standard output (stdout or System.out) in addition to the messages specified above. This will be ignored during the marking process.

Content

Your Content server must satisfy the following requirements:

- It must accept 2 command line arguments:
  - Content-port
  - Contentfile-name
- The command line arguments are interpreted as follows:
  - Content-port is the port number which your server will listen,
  - Contentfile-name is the file describing the items id and the content.
- If more than one arguments are given, your process must print the following message to standard error and then exit:

  Invalid command line arguments for Content\n

  Potential problems include: content file does not exist.

- When Content starts up it must start listening on its port. If it is unable to do so then the process must print the following message to standard error and exit:

  Content unable to listen on given port\n

  If Content is able to start listening on that port, then it should print the following message to standard error:

  Content waiting for incoming connections\n

- The Content server needs to register with the Name Server. It should use the name "Content" with the port and IP. If the registration fails, your process must print the following message to standard error and then exit.

  Content registration with NameServer failed\n

- When your Content starts up, it reads the content file into an internal data structure. It is a very simplified description of stock content. It describes 10 items which can be purchased by the Client. Each of the 10 items is described by:
    - item-id (integer value),
    - item-content (binary data of 200 bytes).

- When Content receives a request from Store, it returns the content with the item id.

Client

Your Client must satisfy the following requirements:

- It must accept one command line arguments:
    - request

- The command line arguments are interpreted as follows:
    - request is a number from 0 to 10 with the following meaning:
        - 0 means that a list of items will be requested from Store
        - number between 1 and 10 means that an item with this item-id will be purchased from Store

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

  Invalid command line arguments\n

- Client first contacts NameServer to get the IP address and port number of Store. If the Client cannot contact NameServer at the specified address and port number, it should print the following message to standard error and exit:

  Client unable to connect with NameServer\n

- After Client gets the address of Store from NameServer, it tries to connect to Store. If it is unable to connect, then the Client should print the following to standard error and exit:

  Client unable to connect with Store\n

- If Client is able to connect with the Store server it should send a request to the Store server. This request is determined by the request command line argument. The request is one of the following :
    - a request for Store to list the items in stock-file
    - a buy request. This includes the item-id to purchase and a credit-card-number (any 16 digit number - assigned in Client).

- The Client should then output the response from the Store. Three responses can be returned by the Store. These are:
    - Response to the list request: item-id and item-price for each of the ten items in the stock-file should be printed to standard output.
    - Response to the buy request: If the purchase is successful, the Store will send item-id, item-price and item-content. These should be printed to standard out as:

      item-id ($ item-price) CONTENT item-content\n

- If the purchase is unsuccessful, the Store will send item-id and the string "transaction aborted". These should be printed to standard out as:

  item-id "transaction aborted"\n

- After sending its response, Store will close the TCP connection to Client. Client should then exit.
- Client must not output anything to standard error other than the messages given above (if appropriate).
- Client must not output anything to standard output other than what is specified above.

Hint

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

Assessment Criteria for Part B

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.

NameServer (10 marks)

- Code compiles successfully (1 marks)
- 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 marks)
- Server correctly prints message expected when listening (1 marks)
- Server correctly deals with registration request (3 marks)
- Server correctly deals with lookup request (3 marks)

Store (40 marks)

- Code compiles successfully (1 marks)
- Server correctly deals with invalid number of command line arguments (1 marks)
- Server correctly registers to NameServer (1 marks)
- Server correctly deals with being unable to listen on given port number (1 marks)
- Server correctly prints message expected when listening (1 marks)
- Server correctly deals with /Client's quote request (10 marks)
- Server correctly deals with Client's buy request (15 marks)
- Server can serve multiple requests at a time (10 marks)

Bank (15 marks)

- Code compiles successfully (1 marks)
- Server correctly deals with invalid number of command line arguments (1 marks)
- Server correctly registers to NameServer (1 marks)
- Server correctly deals with being unable to listen on given port number (1 marks)
- Server correctly prints message expected when listening (1 marks)
- Server correctly deals with /Store's request (10 marks)

Content (15 marks)

- Code compiles successfully (1 marks)
- Server correctly deals with invalid number of command line arguments (1 marks)
- Server correctly registers to NameServer (1 marks)
- Server correctly deals with being unable to listen on given port number (1 marks)
- Server correctly prints message expected when listening (1 marks)
- Server correctly deals with /Store's request (10 marks)

Client (20 marks)

- Code compiles successfully (1 marks)
- Client correctly deals with invalid number of command line arguments (1 marks)
- Client correctly deals with connection to NameServer to get Store's address (1 marks)
- Client correctly deals with being unable to connect to Store (1 marks)
- Client correctly deals with /a quote request (6 marks)
- Client correctly deals with /a buy request (10 marks)

Submission Instructions

The assignment (Part A and Part B) is due at 2pm on Tuesday April 28 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 21 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.