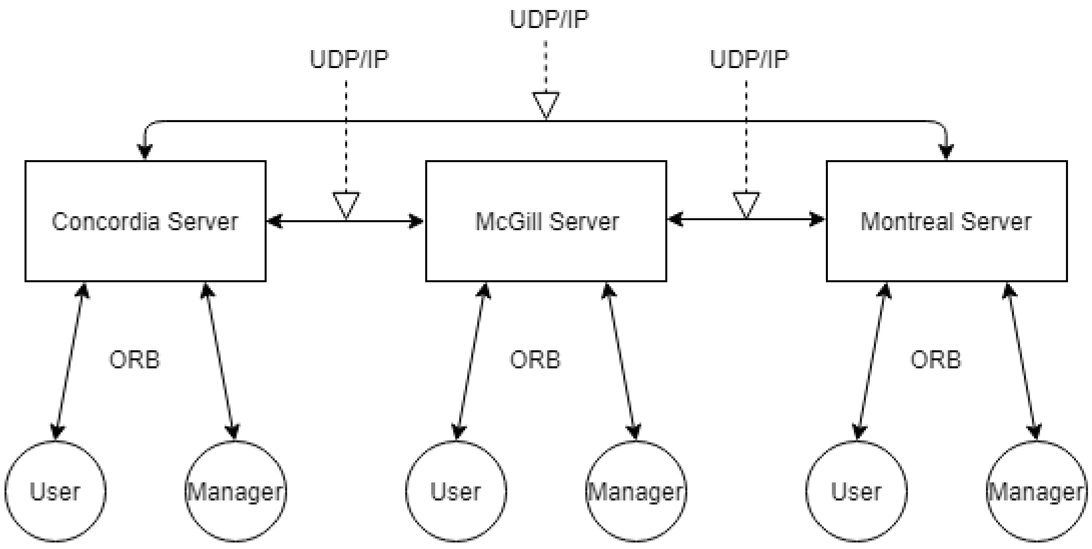
Assignment 2

Techniques:

In this assignment I have used Common Object Request Broker Architecture (CORBA). CORBA is a standard architecture for distributed objects system. It is designed to facilitate the communication of heterogeneous environment, where objects can be implemented in different programming language and/or deployed on different platforms.



There are a couple of things to note about CORBA architecture and its computing model:

- A CORBA-based system is a collection of objects that isolates the requesters of services (clients) from the providers of services (servers) by a well-defined encapsulating interface.
- All requests are managed by the ORB. In other words, every invocation (whether it is local or remote) of a CORBA object is passed to an ORB.

The CORBA System is independent of:

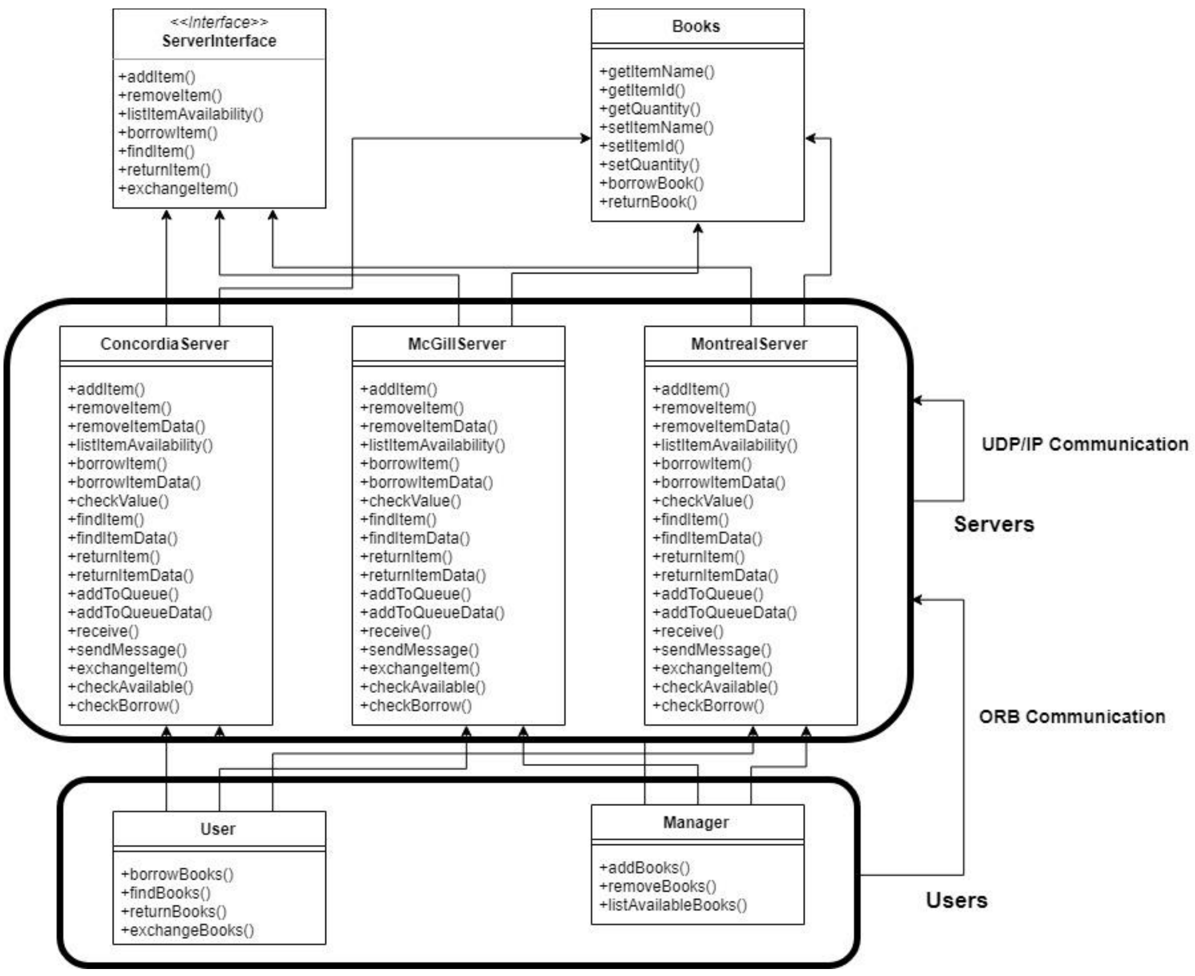
- Programming language
- Hardware platform
- Operating system

In my system object could act as a client, server or both. It is implemented with the help of Interface Definition Language (IDL). IDL describes an interface in language independent way. Object Request Broker (ORB) mediates the communication between client and server, and UDP/IP mediates the communication between server to server. When we compile the IDL interface using IDLJ complier using command **idlj –fall ServerInterface.idl** command it creates 6 java files.

1) _ServerInterfaceStub.java: It is a client side stub file which interfaces with the client object.
2) ServerInterface.java: It is actual interface with characteristic of CORBA interface.

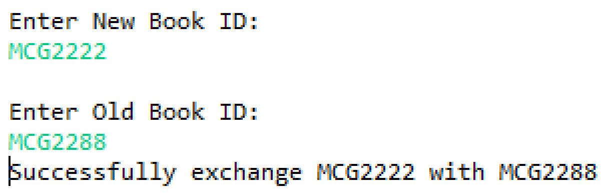
3) ServerInterfaceHelper.java: It provides additional functionality which needed to support a CORBA object in the reference of the Java language.
4) ServerInterfaceHolder.java: It contains reference of an object that implements interface.
5) ServerInterfaceOperations.java: It is a java interface file equivalent to idl interface file.
6) ServerInterfacePOA.java: It is a server side skeleton combined with portable object adapter.

Different servers identifies by their name stored in naming service name. Here in our application there are 3 servers identifies by their associated name in naming service like Concordia, McGill and Montreal. So, client looks for a reference object with associated name in naming service. This is also known as resolving the object name.



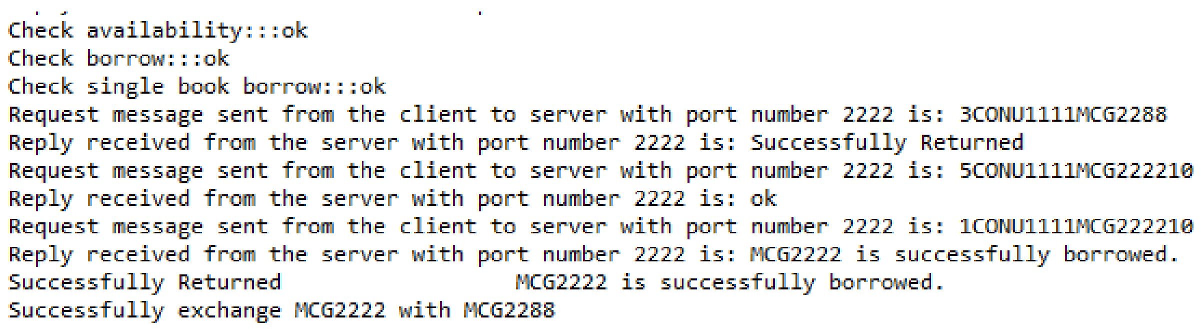
Scenarios:

1) **User tries already borrowed a book from other server and tries to exchange the same book from the same server.**



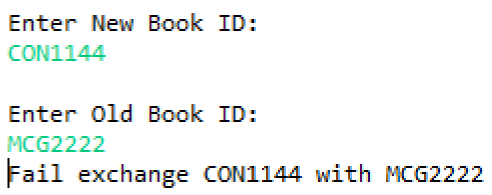
```
Enter New Book ID:
MCG2222

Enter Old Book ID:
MCG2288
Successfully exchange MCG2222 with MCG2288
```



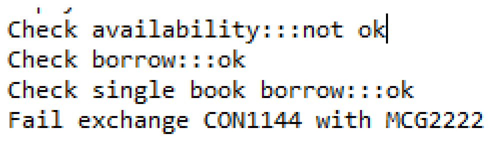
```
Check availability:::ok
Check borrow:::ok
Check single book borrow:::ok
Request message sent from the client to server with port number 2222 is: 3CONU1111MCG2288
Reply received from the server with port number 2222 is: Successfully Returned
Request message sent from the client to server with port number 2222 is: 5CONU1111MCG222210
Reply received from the server with port number 2222 is: ok
Request message sent from the client to server with port number 2222 is: 1CONU1111MCG222210
Reply received from the server with port number 2222 is: MCG2222 is successfully borrowed.
Successfully Returned                    MCG2222 is successfully borrowed.
Successfully exchange MCG2222 with MCG2288
```

2) **User again tries to exchange the book from other server which is not available so atomicity fails.**



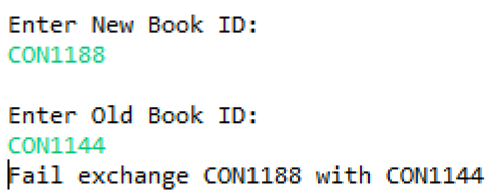
```
Enter New Book ID:
CON1144

Enter Old Book ID:
MCG2222
Fail exchange CON1144 with MCG2222
```



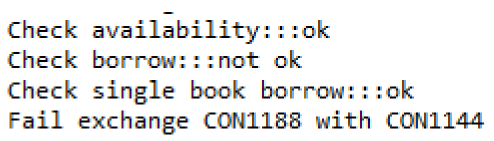
```
Check availability:::not ok
Check borrow:::ok
Check single book borrow:::ok
Fail exchange CON1144 with MCG2222
```

3) **User again tries to exchange with the unborrowed book so atomicity fails.**



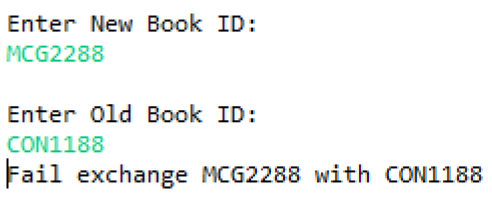
```
Enter New Book ID:
CON1188

Enter Old Book ID:
CON1144
Fail exchange CON1188 with CON1144
```



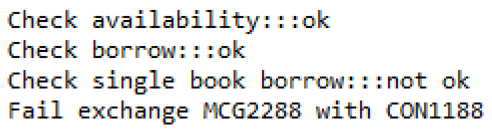
```
Check availability:::ok
Check borrow:::not ok
Check single book borrow:::ok
Fail exchange CON1188 with CON1144
```

4) **User already borrowed a book from other server and tries to exchange a book from the same server.**



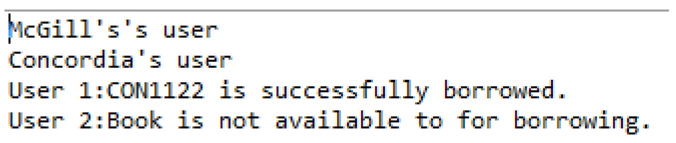
```
Enter New Book ID:
MCG2288

Enter Old Book ID:
CON1188
Fail exchange MCG2288 with CON1188
```

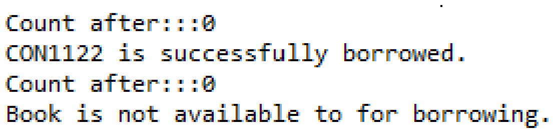


```
Check availability:::ok
Check borrow:::ok
Check single book borrow:::not ok
Fail exchange MCG2288 with CON1188
```

5) **Two users try to borrow the same book which has only 1 quantity at a same time.**



```
McGill's's user
Concordia's user
User 1:CON1122 is successfully borrowed.
User 2:Book is not available to for borrowing.
```



```
Count after:::0
CON1122 is successfully borrowed.
Count after:::0
Book is not available to for borrowing.
```

Most difficult/important part:

The most difficult part in this assignment was:
[1] Problem: To make a system concurrent.
Solution:
To deal with this problem I used **synchronized** keyword in every method which allows only one request at a time in multithreaded application.

[2] Problem: To deal with atomicity of exchangeItem() operation.
Solution:
To deal with the atomicity make an algorithm:

- Check availability of new item
- Check old book is borrowed by the user or not
- Check if new book is from other server then check if it has already borrowed a book from that server or not and, if new book and old book are from same server then allow.
- If all three checks are ok then perform return old book and borrow new book operation.

References:

[1] https://stackoverflow.com/
[2] https://www.draw.io/
[3] https://www.geeksforgeeks.org/
[4] https://www.youtube.com/
[5] https://docs.oracle.com/
[6] https://en.wikipedia.org/
[7] Text book and the slides from Prof. M.L. Liu, California Polytechnic State University