School of Computer Science, McGill University

COMP-512 Distributed Systems, Fall 2025

Getting Started

The following is a description of the steps to run the template programming assignment code (`Template.tar`). You should get started as soon as possible, as there may be issues to work out before you can start the program implementation.

Distributed Components

The programming assignment involves 2 separate components:

- The server host - machine on which the remote server and its registry are running
- The client host - machine on which the client is running

You should be able to use the following machines located in Trottier building for your programming assignment (note that some may be offline or have crashed - you must check which ones are available).

https://www.cs.mcgill.ca/docs/resources/

We recently checked the open-gpu* and tr-open* machines and most were up and running. We recommend that use the `tr-open*` machines for facilitating a consistent environment.

RMI Registry

Before you can start the server, you must make sure the rmiregistry is running on your machine. Start it with the following command:

```
$ rmiregistry -J-Djava.rmi.server.useCodebaseOnly=false 1099 &
```

Note that you have to start the rmiregistry on any machine you intend to use as an RMI server. For convenience, a script `Server/run_rmi.sh` has been provided with the above command. Also see **Port Conflicts** section below.

Creating a Server

First, download and extract the template code (`Template directory` on myCourses) to your working directory. Before the template will compile, you must first make 1 change to the RMI object name:

```
$ cd Server
$ vim Server/RMI/RMIResourceManager.java
```

Edit the line after the TODO note, replacing `group_xx_` with `group_<YourGroupNumber>_`, or any other unique identifier. This will avoid conflicts with other unrelated RMI objects. Next build the code, invoking make:

```
 $ make
javac Server/RMI/*.java Server/Interface/IResourceManager.java Server/Common/*.java
```

Invoking `make` will build the server (for more details, view the included `Makefile`). To run the server, we execute one last command, where the parameter specifies the RMI object name (this will be convenient for running multiple `ResourceManagers`, for more details refer to the script contents):

```
 $ ./run server.sh Resources
'Resources' resource manager server ready and bound to 'group_<YourGroupNumber>_Resources'
```

If you get an exception at this point, it means that you have a problem with paths or executable names. If you are having trouble with permissions, ensure that the `class` and `jar` files are world- readable (permissions 704) and all the directories along your path are world-executable (permissions 705). The rmiregistry and client will get the proxy files from this directory.

For your programming assignment, take note of the following files:

- `Server/Interface/IResourceManager.java`: interface definition for the `ResourceManager`
- `Server/Common/ResourceManager.java`: core (communication independent) implementation of the interface that manages the local state
- `Server/RMI/RMIResourceManager.java`: RMI specific `ResourceManager` implementation that subclasses the common version

Creating a Client

We provide you with the implementation of an interactive client (thanks to Beibei Zou, Chenliang Sun and Nomair Naeem). Information on how to use the client can be found in `clientUserGuide.pdf` available on myCourses. Before building, we must first edit the RMI client implementation to match the server as done above:

```
$ cd Client
$ vim Client/RMIClient.java
```

Edit the line after the TODO note, replacing `group_xx_` with the identifier you chose when configuring your server. Next build the code, invoking `make`. You will see something similar to below:

```
 $ make
make -C ../Server/ RMIInterface.jar
make[1]:  Entering directory '../Server'
Compiling RMI server interface
javac Server/Interface/IResourceManager.java
jar cvf RMIInterface.jar Server/Interface/IResourceManager.class
added manifest
adding:  Server/Interface/IResourceManager.class(in = 1180) (out= 473)(deflated
59%)
make[1]:  Leaving directory '../Server'
javac -cp ../Server/RMIInterface.jar Client/*.java
```

You can see that we assemble a `jar` file of the RMI interface. This allows the client to know about the server interface without the implementation details. For information on the `jar` creation process or the RMI specific details, see the `Makefiles` of both the client and server. Note that while the code goes to the server directory, the interface file could also be copied into the client directory. The client only needs the interface file, not the actual implementation.

To execute the client, we execute one last command, where `<server_hostname>` is the machine running the server. If it is running on the same machine, simply specify `localhost`.

```
 $ ./run client.sh <server_hostname> Resources
Connected to 'Resources' server [localhost:1099/group_<YourGroupNumber>_Resources]
```

Did it work? Congratulations! Now you are ready for the real stuff!

Port Conflicts

If multiple project groups are using the same server, it is possible that you might run into conflicts on using the same, default RMI registry port of `1099`. If you suspect this is happening, you might want to choose a different port number (something like `30XX`) where `XX` is your group number. You can then replace `1099` with your chosen port number in the following files.

`Server/Server/RMI/RMIResourceManager.java`
`Server/run_rmi.sh:rmiregistry`
`Client/Client/RMIClient.java`

Of course it is always a good idea to change it in the beginning itself, before someone else starts using the same server and you start having problems half way into your development.