School of Computer Science, McGill University
COMP-512 Distributed Systems

Group Communication Library (GCL) Documentation

This document gives a high level description about the GCL functionality given to you, which you will be using to construct your Paxos module.

1 GCL Characteristics

The GCL library given to you provides support for both point-to-point communication between two processes as well as the ability to multicast or broadcast a message to all the members in the group.

The notion of a process (identifier) in the context of our GCL is basically a combination of host name and port number, as in `hostname:port`. In our simple GCL library, a java program (process) can only be part of a single GCL group. (I.e., there is no explicit concept of a group name, etc.).

The library itself is built over TCP, and provides a reliable (subject to both processes not having terminated/failed, etc) and ordered (FIFO) communication paradigm.

The FIFO ordering is maintained both within point-to-point, multicast and broadcast messages as well as across the three messaging APIs. For example, if a process p_i sends a point-to-point message m to a process p_j, followed by p_i broadcasting m', then at p_j, m will be delivered before m'. As in conventional FIFO, there is no ordering guarantees on how messages from two separate processes are delivered at any process p_j.

The implementation is thread-safe.

2 API

2.1 Constructor

The GCL functionality is provided by the class `comp512.gcl.GCL` and has the following *constructor*:

```
GCL(String myProcess, String[] allGroupProcesses, GCDeliverListener GCListener,
Logger logger)
```

We will ignore the `GCListener` argument for this work.

`myProcess` is basically the identifier of "this" process, and should be the host name of the server from where the program is being run and the port that it should use. Represented as a single string of the form `hostname:port`.

`allGroupProcesses` is basically a comma separated string consisting of all the processes that forms the group (including the process that is **myProcess**). It will thus have the following format.

`hostname1:port1,hostname2:port2,...`

Ordering of hosts is irrelevant. If you are using the same host for multiple players, then make sure that their port numbers are different. Otherwise they can be the same.

`Logger` is an object of type `java.util.logging.Logger`. It is used by the GCL to log different events (will be very handy for debugging). Please refer to the Java documentation for it's API usage.

The constructor will throw an `UnknownHostException` or `IOException` if you passed an incorrect host name or if it is not able to find the IP address for it.

It can also throw an `IllegalArgumentException` if it does not see `myProcess` also included in `allGroupProcesses`.

2.2 Point-to-point Communication

```
void sendMsg(Object msg, String destProcess)
```

IMPORTANT:-This is a non-blocking send. I.e., it will return immediately as the message is queued internally and processed for sending by another thread. Therefore, remember that because this API returned **DOES NOT** mean that the message is already delivered on the other side.

This API call is used to send point-to-point messages. In this case it will send the object `msg` to the process `destProcess`.

`msg` can be any Java object, as long as it is serializable (marshalling). So if you will be constructing your own objects for messaging, ensure that it implements the right interfaces to ensure serializability.

`destProcess` must be one of the processes in the group, as included (exactly) in the `allGroupProcesses` to the constructor.

2.3 Broadcast

`void broadcastMsg(Object msg)`

This API can be called to broadcast a message to all the members in the group.

IMPORTANT:-This is a non-blocking broadcast.

Internally, it sends point-to-point messages using `sendMsg`.

2.4 Multicast

`multicastMsg(Object msg, String[] processess)`

Used to send `msg` to a subset of the processess in the group.

IMPORTANT:-This is a non-blocking multicast.

each process in `processess` must be already present in `allGroupProcesses` when the GCL was created.

Internally, it sends point-to-point messages using `sendMsg`.

2.5 Reading messages

`GCMessage readGCMessage()`

To be called to retrieve a message that is received by the GCL of this process. Messages are queued internally and made available when the read is called. If there are currently no messages in the GCL buffer incoming queue, **this call will BLOCK**. Note that the API **DOES NOT** on its own distinguish between whether a message was sent through `sendMsg`, `multicastMsg` or `broadcastMsg`. If you need such capabilities, you will have to add it in your layer.

Will throw an `InterruptedException` on a thread that is blocking on this call (waiting for new messages) if the GCL is being shutdown.

`comp512.gcl.GCMessage` is a simple class that consists of the following two public members.

`String senderProcess` → This is the process that sent the message. Format is also of the type `hostname:port`.

`Object val` → This is the message object that was sent (using one of the messaging APIs discussed above).

2.6 Shutting down the GCL

`void shutdownGCL()`

This will result in the GCL library shutting down its internal communication channels with the various processes in the group. The function call returns once it receives a similar

confirmation from all the non-failed members in the group (or in some cases might just time out and return). It is possible that some of the messages that are still in the GCL internal buffers may not get delivered to the application layer or despatched to the destination process (depending one the scenario) once a shutdown is initiated.

Making a call to any of the APIs after a GCL shutdown is initiated will result in them throwing an `IllegalStateException`.

2.7 Misc

The GCL layer will try automatic re-transmission of messages up until its own process is shutdown. This is applicable, say, if the receiving process has not yet been started or has failed, etc. In the former case, it will receive the messages once it is online (and its GCL layer has started).