UMEÅ UNIVERSITY, DEPARTMENT OF COMPUTING SCIENCE

GCom project (60 points)

Distributed Systems - 5DV147

1 INTRODUCTION

A distributed system is composed of group(s) of processes that coordinate their activities by exchanging messages over a network. For such systems, pair-wise exchange of messages need not be the most efficient or reliable way to communicate. *Multicast* is an operation that sends a message from one process to a group of processes. This is typically achieved without the sending process having explicit knowledge of the group members. Common applications include, but are not limited to:

- *Replication-based fault-tolerance*: A service is replicated across multiple servers forming a group. Client requests are processed by all servers. In this setup some servers can crash without affecting the availability of the service.
- *Ad-hoc network service discovery*: Multicast messages are used by clients to discover, and register to, services.
- *Replication for high-performance data access*: Commonly used data objects are replicated for performance improvements. Updates to data replicas are propagated using multicast-messages.
- *Propagation of event messages.*

A *middleware* is a software layer that abstracts and hides the heterogeneity of underlying networks, operating systems, and programming languages. Examples include CORBA [DS5: 8.3] and Java RMI [DS5: 5.5], although the latter merely supports a single programming language. In addition to solving the heterogeneity problem, a middleware provides a model for distributed communication, e.g., *remote method invocation* and *remote event* *notification*. CORBA and Java RMI implement the former approach, i.e., a program can invoke a method in an object located on another computer.

The goal of this project is to design and implement **GCom**, a middleware for group communication. The middleware must implement some API that in turn can be used by programmers to develop applications that make use of reliable messaging. GCom must support various types of multicast, guarantee causal message ordering and handle group membership issues.

Group membership management is a complicated problem and has been the focus of much research. For this project, we will accept simplified solutions to the group membership problem and instead focus on the multicast types and message ordering algorithms.

The project must be solved by groups of two students.

2 Purpose

The overall aim of this project is to increase your knowledge of distributed systems and distributed programming. More specific goals include:

- *Understanding of different communication types*, their characteristics and scenario suitability.
- *Understanding of causal message ordering algorithm*, its characteristics and effects.
- *Understanding of basic fault tolerant techniques*, with focus on maintaining robustness and availability of a system under network partitions and unexpected events.
- *Understanding of system architecture design impact*, with special focus on differences between centralized and distributed solutions.
- *Development of software engineering skills*. Practice the ability to convert a loosely specified problem into a concrete design and implementation.
- *Development of written communication skills*.

3 Description

The GCom middleware consists of three (logical) modules, the *group management module*, the *communication module*, and the *message ordering module*. These are, respectively, responsible for handling group membership issues, communication message exchange semantics, and message (re)ordering issues. All of these modules need to function properly in order for your system to be able to ensure correct message delivery semantics.

3.1 Group management module

In order to send messages to the processes in a group, GCom must keep track of the members of a group. This can be done either *statically* or *dynamically*.

In a static group the structure of the group is predefined and only group members can join the group. Until all members have joined no member can send data messages to the group. Once the group is established, message sharing can begin and no new members can join (or re-join) the group. The group management module must however be able to handle that members may leave static groups at any time (due to crashes).

In contrast, in dynamic groups members may join and leave the group at any time. Processes that re-join the group may be handled as any other joining process.

The four main tasks of the group management module are:

- To *provide an interface for group management*: The group management module provides operations to create and remove groups, as well as to add and remove members from a group.
- To *detect errors*: The module monitors a group and indicates when a member of the group crashes (or for some other reason become unreachable).
- To *notify changes in group membership*: The module notifies all group members about changes in group composition.
- To *resolve group names*: When a process sends a message to the group, the group management module resolves the group name into a list of group members.

Consider the following issues when designing the group management module:

- *Scalability*, e.g., is it suitable to keep a complete list of all groups in each process?
- *Limitations*, e.g., can your group management module handle all types of errors? What can the system not guarantee?
- *Partitioning*, what happens if the group is partitioned, e.g., due to a network segmentation?

Recommended reading: DS5: 15.4, 18.2.2

3.2 Communication module

Processes in a group can communicate via various mechanisms. The creator of a group specifies which communication mechanism to use for all data messages in that group. GCom must support the following mechanisms:

- Basic **non-reliable** multicast.

- Basic **reliable** multicast (optional but you will learn a lot if you implement this feature).
- Tree-based **reliable** multicast **(bonus level)**.

For the tree-based multicast algorithm, read about *multicast routing*, e.g., in [DK: 4.7.2]. For reliability issues regarding tree-based multicast see, e.g., IP-multicast described in [DS5:15.4.2]. The requirements for the reliable tree-based multicast are simply as follows:

- It must be **reliable** (see formal definition in the book).
- Messages must be sent along a path in a tree structure such that a node only sends to its children or parent (i.e., only nodes it is connected to directly).
- If you attempt to solve this level, you **must** come by our office and explain your approach. Bring your idea in writing (images are very useful for this as well). This is to avoid misunderstandings that end up costing your group important time that would be better spent testing and maintaining your code.

Recommended reading: DS5: 15.4, DK: 4.7.2

3.3 Message ordering module

Various types of delays in networks and computers may cause reordering of messages in a process, e.g., the reply to a certain request may, in some process, arrive before the request. For some applications this type of behavior is unacceptable. The most common ordering requirements are:

- FIFO: messages from a process should be delivered in the order that they were sent.
- Causal: the ordering captures causal (cause and effect) relationships via happened-before ordering.
- Total: messages may be delivered in arbitrary order as long as all processes in a group deliver the messages in exactly the same order.

Hybrid message orderings add extra ordering requirements. For example, in *Causal-Total* messages are first sorted according to causal ordering then according to total ordering. Total ordering combined with reliable multicast is also referred to as *atomic multicast* and is often used for replication and transactions.

The GCom middleware must be able to deliver messages according to **unordered** (i.e., as they arrive) and **causal** orderings. Note that GCom must be able to deliver messages using either ordering and all types of communication mechanisms that you implement, basic non-reliable, basic reliable multicast, and tree-based reliable multicast if you aim for the bonus points.

Recommended reading: DS5: 15.4.3

3.4 Testing and Demonstration

In order to ensure the correctness of the GCom middleware, you must implement a chat application. You must also implement a debug application that allows you to simulate dropped, rearranged, or delayed packages, as this will allow you to test whether your message ordering module and reliable multicast modules are working correctly.

Note that it is perfectly fine to let the test and debug application be one and the same application, although it is nicer from a software engineering point of view if you keep them separated logically. You have to agree that regular users do not want to get a full debugging view when they start a program, so if you make a combined test and debug application, at least keep the functionality separated so the debug view can be enabled upon request, rather than always enabled. This may also be helpful in light of the following project.

Your test application will (together with your debug utility) be used during your practical demonstration of this project. It is hence paramount that you not only implement GCom, but also *clearly* are able to demonstrate that your implementation is correct. This should be obvious, as you probably want to convince yourselves that your solution is correct before attempting to convince the teachers.

The debug utility and test application (combined) must *clearly* demonstrate *at least*:

- *That messages are delivered according to the specified ordering*
- *How messages are propagated in the network*: Show both the path a message takes and how many times a certain process has received a certain message.
- *The content of hold-back queues and other buffers*: Present all messages waiting to be sent or delivered as well as values of vector clocks and other counters.
- *Current system performance*: As a measure of the system performance, count the number of messages (including control messages) required to perform an operation (send one message with a specific ordering and certain multicast).
- Also, for clarity reasons, *the test and debug application must feature graphical user interfaces*, as the amount of information that needs to be presented exceeds what is reasonable for a text-only user interface.

3.5 Tools

You are free to use any tools you want for this project, in fact, selection of appropriate tools is considered part of the assignment. You will however have to motivate your choice of tools with respect to, and in terms of, what you expect to learn from this course and project. If you find a tool (there exists quite a few) that solves the assignment with little or no effort from your side, you are naturally **not** allowed to use it (one such tool is for example

JGroup[1]). You are of course allowed to use documentation and research papers from existing solutions when designing GCom. As middleware-development is part of the course focus, you are not allowed to implement GCom using TCP/IP sockets.

More specifically:

- You may implement GCom in any programming language and middleware of your choice (as long as the rule about non-trivial solutions is not broken), however, the teachers will not provide help for other systems than Java using Java RMI.

- Make use of any IDE and/or plug-ins you feel makes the job easier. You will write a quite substantial amount of code (in Java, approximately between 2000 to 7000 lines including the GUI). Use tools that facilitate this.

- Since your test and debug application(s) must feature graphical user interface(s), use a suitable tool for interface construction. There are IDE plug-ins for this task, and you will likely save at least half a day if not more by using these tools correctly.

3.6 Advice

Some helpful hints:

- Make the design and implementation of GCom modular to simplify the combination of message orderings and multicast types.

- Consider how debugging will work early on in the design process. This is very important! Retrofitting debug functionality onto your application may require a non-trivial effort.

- Integrate testing and implementation. Test modules separately as they are implemented. Early unit testing makes it easier to detect and limit effects of bugs. Also perform integration testing to ensure correctness in inter-module communication.

- Read the algorithms in [DS5] carefully and consider what data structures are required to implement them. The algorithms in [DS5] are described using a high level of abstraction, and cannot be implemented as is.

- The fact that a system cannot be formally proved to work does not make it impossible to implement - consider for example the Internet. Read [DS5: pages 659 and 668-670].

- Analyze (and reanalyze) your design before you start implementing it! You will most likely not have enough time to comfortably make major redesigns as the weeks have gone by.

[1]http://jgroup.sourceforge.net/

3.7 Detail list of project requirements

The mandatory features that must be implemented and their value in points toward the project final score are listed below.

Feature	Points
Group management for dynamic groups	20
Basic non-reliable multicast	5
Basic reliable multicast	optional
Message ordering algorithms	10
Chat application	5
Debug application that clearly demonstrates the correctness of GCom	5
Implementation points	45
Written report	15
Total points of project assignment	**60**

In addition, you must demonstrate your solution to us and hand in a complete report fulfilling the requirements as described below.

3.8 Bonus Points

Everything on the mandatory part + Tree-based reliable multicast. In order to obtain the bonus points, all features from the mandatory part **must** work correctly and you must score at least 50 points on the theoretical part of the evaluation (from the written assignments). We will add 20 points to your score if you implement this feature completely.

If you decide to work on this part of the project, make sure that you and your partner have the same ambitions and goals for this project. If you attempt to solve it, you **must** come by our office and explain your approach. Bring your idea in writing (images preferred). This is to avoid misunderstandings that end up costing your group important time that would be better spent testing and maintaining your obligatory code.

4 Deliverables

As an aid in planning your work, we divided the project into two deliverables. Read both the specification and suggested literature **carefully** before starting working on deliverable 1.

4.1 Deliverable 1 - Analysis and Design (Optional)

This deliverable is optional but we encourage you to do it since we can offer you feedback about your overall plan and design. If you decide to work on this deliverable, you must include at least the following sections:

- *Analyze the problem.* What should be done, and what problems need to be solved? Formulate a requirements specification for GCom, based on the problems you identify. What *must* be implemented, and what *may* be implemented, time permitting? See [RFC 2119][2] for suggestions on how to write such sections.

- *Identify suitable tools.* Are there any third party components that could be used? Learn the benefits and drawbacks of possible choices by implementing simple test applications before choosing a tool that you will devote energy into using. The sooner you learn the limitations of a technology you use, the better.

- *Plan your project* Write a project plan, complete with milestones (dates when you will have completed parts of the project) and division of work. You will be required to refer back to this in your final report, and discuss how well your project followed the project plan.

- *Design a solution.* Write a design (as detailed as possible) for your solution. Try to walk through the conceptual parts of the design to convince yourself that it will be feasible when you implement it. Having a sound, modular design is important. Your design should be detailed enough to not need any major additions during implementation. Minor modifications are allowed of course - even the most carefully designed system may require changes due to e.g., unforeseen properties of the tools used.

4.2 Deliverable 2 - Implementation and full report

Please note that you can *base* the full report on the first analysis and design document, which means that a good first report will make the full report easier to complete!

It is a **strong** requirement that your system can easily be started from a normal Unix terminal. If you have a complicated command line, please include the appropriate scripts for each component. See this basic scripting tutorial[3] for a quick guide on scripting.

5 Hand-ins

Deadline for handing in solutions are found in the course schedule[4]. Keep in mind the policy for late submissions.

The project consists of two or three hand-ins:

1. Written hand-in of deliverable 1.

2. Written hand-in of deliverable 2 (complete report) and implementation.

3. Demo of your system.

[2]http://www.ietf.org/rfc/rfc2119.txt?number=2119
[3]http://www.linuxconfig.org/Bash_scripting_Tutorial
[4]schedule.html

5.1 Demo

During your demo, you will need to convince the teachers that your implementation works. Bring a test protocol, i.e., a series of tests that **clearly** demonstrates that your GCom fulfills the requirements and a test tool which can be used to apply it. The protocol should include tests for a combination of message orderings and all multicast types that you implement. Bring a copy of the test protocol on paper, see [DS5: 668] for suggested notation. Your test protocol must clearly state your names, user names, and which level you intend to demonstrate.

5.2 Report

A written presentation of your results is an important part of the project. This is true not only for this project, but in virtually any software development process. People lacking the skills required to communicate the results of their work (verbally and in writing) will find it difficult to work in the software industry. The purpose of the report is hence not to make the teachers happy, but to improve your communication skills.

Try to focus on the specific characteristics of this project when you write the report. Do not follow some predefined outline from some previous course for the report, just for the sake of it – at this point in your education, you are more than able to decide which parts are important and which could be left out. As this project emphasizes analysis and investigation of a loosely specified problem, include any assumptions you made during the analysis phase in your report. Also discuss problems encountered and alternative solutions considered in the analysis. The report should also discuss to what extent the requirement list is fulfilled, as well as to which extent you could adhere to the project plan. Finally, remember that GCom is a general middleware, which just so happens to be used by a chat application for the project, this means that it is important to document how one would use it to develop other applications requiring group management and group communication.

Try to maintain a focus on readability while writing the report. Make the description of your system as clear as possible, and consider your report's level of detail. Your report is the first impression users (and teachers) get of your system. Ensure that figures and diagrams are easy to read. Proof-read your report carefully and use spelling-checking tools (for the Latex crowd, use `"aspell -lang=en_US -c filename.tex"`, nice typesetting does not make spelling mistakes less of a distraction), and yes, the report must be written in English.

To make it easier for us to test your implementation, **state the complete command line required to execute your solution on the front page of the report**.

5.3 Final note

Remember, you need to get at least 50 points on the aggregate of all projects to pass the course. Any bonus points from the assignment can only be used to get a higher grade once you pass. Furthermore, any bonus points obtained only count during the three exams offered for the course during the next 12 months, not for later offerings of the course.

6 References

[DS5] Coulouris G., Dollimore J., Kindberg T. and Blair G.: *Distributed Systems - Concept and Design.* Fifth edition. Addison Wesley (2005)

[DK] Kurose J.F. and Ross K.W.: *Computer Networking - A Top-Down Approach Featuring the Internet.* Third edition. Addison Wesley (2005)