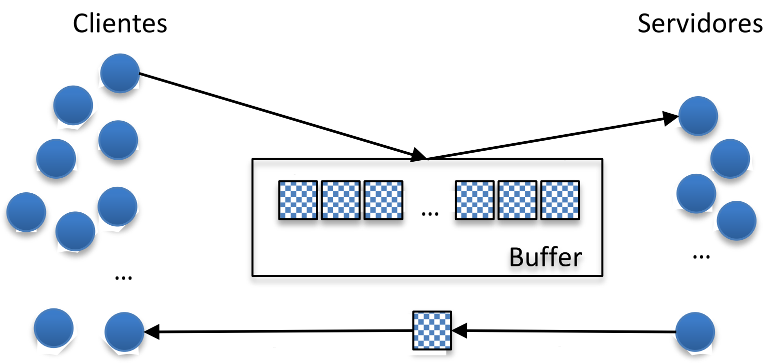
Universidad de los Andes
Systems and Computing Engineering Department
ISIS 2203 Computational Infrastructure

Caso 1
Concurrency Management

We want to implement a system with several servers that run concurrently to answer queries:

- Clients create queries; a query is just a number randomly generated. Clients expect to receive an answer from a server; an answer is the original number increased by 1. A client receives an answer and then reads and discards it.
- Clients do not directly communicate with servers. They do it by using a buffer. This kind of communication allows a system to attend clients independently of number of clients or servers.
- Servers take queries from the buffer, generate an answer and tell the client that sent that query that an answer is ready.
- The following figure presents the communication scheme. In the following section we explain details.



Objective

We want to design a communication scheme to handle queries from multiple clients. In this case, we will use *threads* running in the same machine to represent clients and servers (in a real system clients should run in different machines; but this is only a prototype and we are focusing on challenges related to concurrency).

You must solve the problem using Java and *threads*. To synchronize thread behavior you only may use the following Java instructions: `synchronized`, `wait`, `notify`, `notifyAll and yield`.

System Behavior

A client *thread* creates a query, stores it in the buffer and expects an answer to every. It repeats this behavior a given number of times and then ends. The number of clients and the number of queries per client must be configuration parameters (each client needs to receive a number that will indicate the number of queries to create). To represent a query, a client *thread* will generate an object of Message type, initialize it and then send it. When a client has sent all the queries tells the buffer it is leaving.

A server *thread* runs as long as there are clients; it must end whenever all clients have finished. Server *threads* will answer queries stored in the buffer (to answer means increasing the value by one and telling the associated client that it may continue). The number of server threads is also a configuration parameter.

Number of clients, number of servers, number of queries per client and buffer size must be defined in a file, the main must read the file and run accordingly.

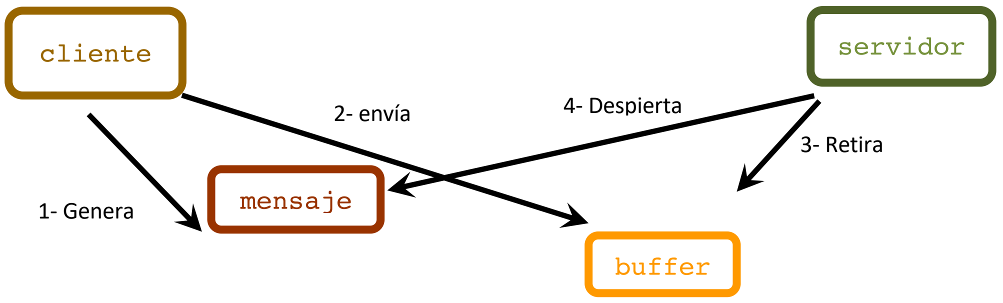
Design:

This system will have: clients, servers, a buffer and messages. The previous section describes clients and servers.

Regarding the buffer, it is a place for clients to store messages (/queries). Servers check the buffer to pick and answer those queries. This buffer has a defined storage space (a defined number of messages) and works following a producer-consumer pattern. In addition, messages are objects that have a client´s query and a server´s answer.

The system follows this protocol: a client creates a message and tries to store it in the buffer. If it is not possible then the client uses active waiting. However, a client must yield the processor after every attempt to store a message. Once it succeeds storing a message, a client must wait for an answer; this time it uses passive waiting on the message object.

Servers try to retrieve a message from the buffer. If a server cannot retrieve a message, it will use active waiting to try again. However, a server must yield the processor after every attempt. Once it succeeds retrieving a message, it generates an answer and notifies the associated client. The following figure shows the protocol.



Client and server must be able to "talk" to the buffer; they do not "talk" to each other. The buffer need to know the number of clients in the system. It does not know the number of messages. Buffer size must be a configuration parameter; this number is not related to client or server numbers, they are independent.

Submission

- You are expected to submit a .zip file with source code and a document (word or pdf) explaining your design and program behavior. In particular, you should explain for every pair of object that interact, what mechanism you used to synchronize their behavior. Also, the global behavior of the system. The name of the file must follow this structure: caso1_*login1*_*login2*.zip
- We will work in groups of 2 people. Groups should not show their solutions to other groups. Groups of 3 are allowed only if they have two students from Colombia and one student from New Zealand.
- We expect all members of a group to participate in the building of the solution.
- The solution must be uploaded to Sicua+ by one of the students (please coordinate who will be in charge of this task). The document must include name and id number of all the students in the group. Please, do not forget to include all the members of the group as modification to this will affect your grade (the grade will be divided to consider the member that was not reported).
- **Deadline: February 20th 23:55 (p.m.)**