FELIPE MATHEUS COSTA SILVA
LOGIN: fcosta1
STUDENT NUMBER: 706279

Converting to Asynchronous Implementation: the submitted File Synchronization uses a Synchronous implementation with blocking processes. It references two flux directions: pull (the client receives the instruction) and push (the client sends the instructions). Both Client and Server implementations import the Threads implementations used for sending (FileCheckThread and NextInstructionThread) and receiving (ReceiveNextInstructionThread) so they can auto set up depending on the chosen direction (note that the client always sends the negotiation instruction where the direction is defined).

Converting this implementation to Asynchronous demands two main changes. The first one concerns the way in which the client sends instruction messages: while the NextInstructionThread will still take the instructions from the queue, it should work alongside a middleware that must send all the output from the NextInstructionThread to a buffer on the server side (the middleware must guarantee a reliable delivery). With this approach, all the messages sent by the Client will receive responses automatically and the process is not blocked. When the server is ready to process instructions, all it needs to do is access the buffer.

The second change refers to the way an instruction message is checked. The server can access the buffer, pick the right message and delete it after use, so there is no need to "expecting" or "ack" replies.

Note that this explanation regards the "push" direction, when the client sends instructions to the server. Also, the middleware must have a flow control based on the buffer size. When the buffer is full, the middleware will block the sending process and wait for buffer to be released.

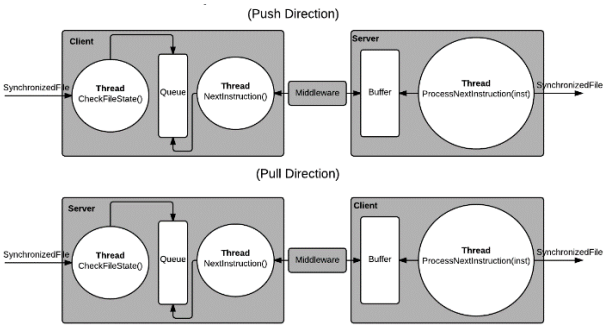

Figure 1: Asynchronous Architetural Model

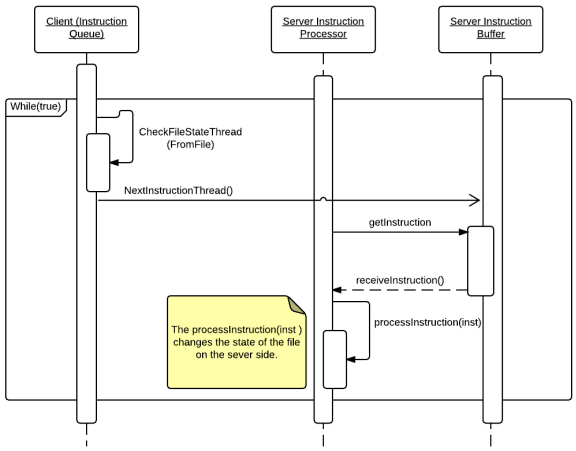

Figure 2: Sequence Diagram (Push direction)

Arbitrary Block Size: in order to allow a receiver (either Client or Server) to work with different block sizes the sender must create and commit a new type of negotiation message (let us call it blocksize message) to each instruction. Thus, the work of receiving and processing an instruction would make necessary a pair of messages.

Every time a sender takes an instruction from the queue, it must calculate the size of that instruction (block plus other instruction details), put this number in a blocksize message and send it. The receiver must use the length contained on the blocksize message to allocate space memory in which the instruction message will be stored. A negotiation message structure could be as follows:

{
"type":"blocksize"
"length": 1024
"counter": 1
}

The sender and receiver both know that a negotiation message precedes an instruction message, being relatively easy to implement this interaction. The synchronous implementation gives you blocking processes that make simple the management of messages sequence.

It is important to highlight that the blocksize message has, is most of the cases, a predictable size. All it takes to receive it always successfully is working with a slightly bigger buffer that accommodate small variations in the size of the message.

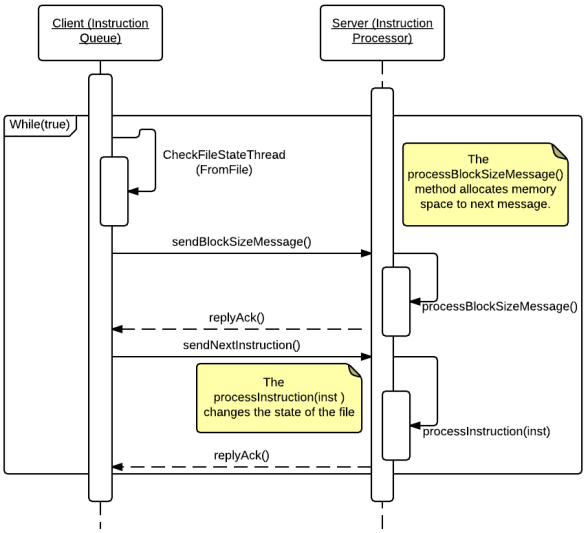

Figure 3: Sequence Diagram to arbitrary block size (UDP).

File Synchronization Systems:

Dropbox: this software is one of the most famous multi-platform file storage and synchronizer available. It works using cloud servers where a version of to the clients' files is stored in order to be spread across different platforms. The client has access to a multiplatform software that, once installed and logged into the user account, will keep track of the file versions. If one client changes a file, it will be pushed to the server and if a new version of the file is available on the server it will be pulled to the client.

As a non open source application, the user does not have complete access to the server and neither have knowledge of the deep layers of the activities performed.

Studies have been perfomed to understand the real operation of Dropbox [1]. What is known, among other things, is:

- Dropbox server deals with data blocks (identified by SHA256 hash values) with size up to 4MB;
- Commit of messages occur parralel to push operations;
- A client TCP connection is kept continuolosly open to notificate the server.

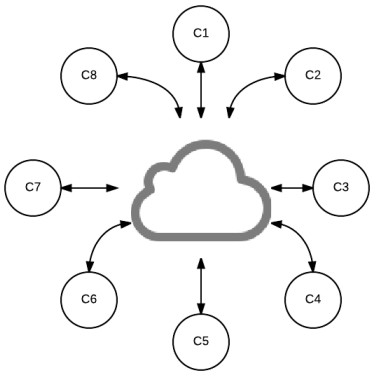

Figure 4: Dropbox architecture: each circle represents a device. A user can have multiple devices.

Unison: this free file synchronization program works providing ways to save files through different computers. These files have their state and contents synchronized, with changes being detected and propagated through the set of computers connected, keeping them always up to date.

Using softwares like Dropbox, in which cloud computing is used, the user does not have any control on the server. By choosing Unison, you have ensured the total control of your files and the way they are handled. Thus, you can always verify if the changes are being made correctly and configure your own server in order to reach personalized features that you may need.

To run Unison, the user need to set up a personal SSH server (with Login and Password). The layout of the computer network may vary depending on the users' necessities, but the most used is the Star setup. To reach this topology, the user must dedicate one machine to be the server, where all the changes will be hold and from where they will be spread. In this case, any changes performed by a client in a file will be sent to the server that spread the update to the other clients. Thus, the client does not communicate changes to other clients, only with the server.

Interruptions, fails and conflicts are handled automatically by Unison as this software keep track of all the clients versions of a file.

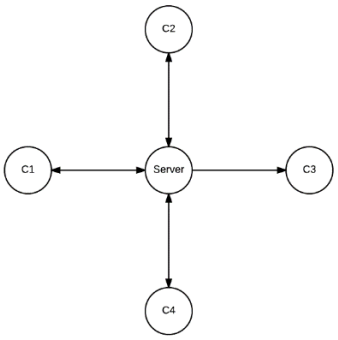

Figure 5: Unison most used architecture.

References:

[1] Drago, I., Mellia, M., M Munafo, M., Sperotto, A., Sadre, R. and Pras, A. (2012). Inside dropbox: understanding personal cloud storage services. pp.481--494.

Markus-gattol.name, (2014). File Synchronization with Unison. [online] Available at: http://www.markus-gattol.name/ws/unison.html [Accessed 21 Sep. 2014].