# AC01: DISTRIBUTED SYSTEMS - A NAÏVE VIEW

A diagram of a process

Description automatically generated

From this diagram we can see the basic design we have followed in our program. The master program has a socket pair with every client server through which they communicate. This master process also has de variable, as such it’s the only one that accesses it directly.

A diagram of a company

Description automatically generated

The communication between a master process and a client one follows the behavior shown above. The master will receive an initial ‘r’ or ‘w’ character. If it’s the former it returns the value of the variable through the channel it received the petition. If it’s a ‘w’, the first two messages will be the same, with the addition that the client will send the updated value for the master to store it. The master will response with a 2 as a confirmation message. Notice that this behavior is the same for all processes.

A diagram of a diagram

Description automatically generated

We can further observe the behavior of the master with these diagrams. It will wait in each socket until it receives one of the two acceptable characters. At which point it will either send the value of the variable back and go on to the next socket or wait for a new value to update the variable. After the confirmation message is sent back, we change the socket as well. This loop iterates through all servers checking every connection.

A diagram of a diagram

Description automatically generated with medium confidence

This last diagram represents a writing client server, where it will send a ‘w’ character to the master before reading the socket until we receive and answer. Once the value is received, we increase it and send it back for the master to update the variable. After that the server executes the sleep instruction and loops through the behavior again for 10 times.

A diagram of a connection

Description automatically generated

The other behavior of a client server is only reading. Where the sent character will be an ‘r’. We will expect the value of the variable from the master before executing the sleep and iterating through the loop again.

## Limitations of our algorithm

The first limitation to consider from our algorithm is the scalability. As we have previously explained, our algorithm works as a loop, going from the first created process to the last. This is achieved by having a socket per process. However, the main problem is that if one of the processes gets stuck, or gets delayed, it affects the following ones. Therefore, if the number of processes grow, the delay does too, and exponentially.

The other limitation we can find is the reliability, due to Single Point of Failure (SPOF). Since we only have 1 master and there is no backup or activity that checks if the server is working accordingly, if the server crashes, the whole system does as well.

## Performance result after increasing the number of processes

Arbitrarily, we have increased the number of processes, in order to check the functionality of our system and see the concurrency of the variable. The first try was, obviously, with 1 process, to ensure that our system worked correctly. Then we incremented it to 4, then to 10, and then to 50, were we saw the first issue. When the variable was at 114 it got stuck, which by our case that pairs update the variable, so 25 pairs are updating it, therefore at around the fifth round of updating it. We were not able to figure out what was causing this issue, maybe a buffering error, or simply that too many clients requesting to the same server is limited by either Matagalls or Linux in general.

## Complication in adding new servers in mid-run

Since the functionality our program is loop-based, as long as there is no element to read from the socket, nothing will happen. It will infinitely wait until the client sends the request to read from the socket. Therefore, in order to skip this loop, or do other things at the same time, it should be added a thread that waits for a new connection to accept it. Once it is accepted, using dynamic memory, our array of sockets should be updated into getting a new element, so we would have to use mutex in order to avoid using the same instance of array of sockets when one is modifying it.