Casey Brooks
CSCE 313 – 504
MP2 Report

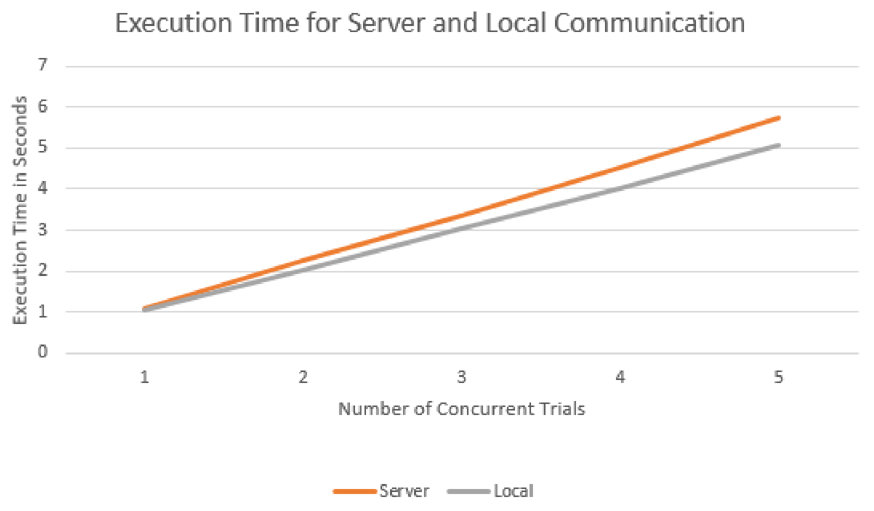
This machine problem sets up a simple client-server local network so that its efficiency can be analyzed, but also so that we get an intuitive understanding of how interprocess communication works. For this problem, most of what was going on was already given to us, so all we had to do was fork the main process from the server so that we could test how the two independent processes work with one another.

The framework for communication between processes here consists of three parts: the server, or the process that typically handles the data; the client, where the data is consumed; and the request channel, which is the link between the client and server. The server, upon receiving a request from the channel, determines what to do with the data, and then returns the appropriate result: in this case, it reads a string and returns another string based on the input.

The RequestChannel class takes a series of requests from the client and funnels them to the server, while checking the channel to ensure that it is only doing one thing at a time: in other words, preventing the client and server from editing the same data at the same time, thus falling out of sync. Because both the client and the server share a common request channel, and because the channel forces one-way communication, there should never be issues with the client and server falling out of sync.

But this extra bit of code that connects the client and server will necessarily slow down the execution of the program as a whole (as compared to the same functions being executed all in one process). To test this, I set up a function that would roughly emulate what the server does (a simple function that takes a string as an argument and returns the same string to the user). Then using the same functions from the first machine problem to measure execution time, I repeatedly ran the same requests to the server and also to the local function, measuring the execution time of each independently and recording the results. At first, I measured the time for forking the client and server processes and testing the communication just once. I did this three times to get an average running time, and then did this again with calling the whole process twice, all the way up to five times. Each time I did the process of forking a child to create the server, except with the local base test I did not communicate with the server. This was done to ensure that the only variable being measured was the time from sending requests to the server to when it receives the result, and nothing about the background processes would skew results. The raw results are displayed below:

	Time for Server			Time for Local			
Trial No.	S	µS	total µS	S	µS	total µS	% difference
1	1	45,772	1,045,772	1	25,319	1,025,319	
	1	141,391	1,141,391	1	9,959	1,009,959	
	1	79,437	1,079,437	1	41,085	1,041,085	
average	1	88,867	1,088,867	1	25,454	1,025,454	6.18
2	2	314,272	2,314,272	2	9,533	2,009,533	
	2	218,207	2,218,207	2	13,823	2,013,823	
	2	229,188	2,229,188	2	5,059	2,005,059	
average	2	253,889	2,253,889	2	9,472	2,009,472	12.16
3	3	319,826	3,319,826	3	10,438	3,010,438	
	3	428,172	3,428,172	3	38,107	3,038,107	
	3	339,687	3,339,687	3	78,699	3,078,699	
average	3	362,562	3,362,562	3	42,415	3,042,415	10.52
4	4	450,307	4,450,307	4	12,261	4,012,261	
	4	816,687	4,816,687	4	51,588	4,051,588	
	4	311,360	4,311,360	4	23,775	4,023,775	
average	4	526,118	4,526,118	4	29,208	4,029,208	12.33
5	5	698,468	5,698,468	5	18,200	5,018,200	
	5	752,008	5,752,008	5	97,835	5,097,835	
	5	793,461	5,793,461	5	125,496	5,125,496	
average	5	747,979	5,747,979	5	80,510	5,080,510	13.14



As is shown, there is an average difference of between 10 and 13 percent in terms of execution time for the server and the local function, meaning that local function runs about 12% faster than the inter-process communication, and the graph shows that as more trials are run in a row, the time that it takes to execute them all grows linearly, but the server time increases slightly faster with more trials.

So in general, communication to a server process will always take longer than if the request were to be handled in the same process. Even in this case, a very simple function that takes a string and returns, the time to execute it is significantly longer than just handling it on its own, and this is when the program is running on the same machine. When this process is being handled over multiple computers on different networks, it is quite easy to see how interprocess communication throttles the speed with which an application can be run and make it difficult to run networked applications well.