ECE 650

Methods and Tools for Software Engineering

Assignment-1

Gursharan Sandhu
Student ID-20635962
Section-1
g27sandh@uwaterloo.ca

UNIVERSITY OF
Waterloo

Introduction:-

This Assignment we have discussed the two ways to solve the Single Producer and Single Consumer problem in the Linux Environment. For the first case, we are using multiple processes with the help of a message queue solution in which two processes are not sharing any memory. The other process involves the use of the threads and a shared global data structure. With the help of this problem, we are able to analyze the performance characteristics of the both approaches and able justify which one has an upper edge over the other one.

Results:-

1. With the Multiple Processes-

Process Implementation

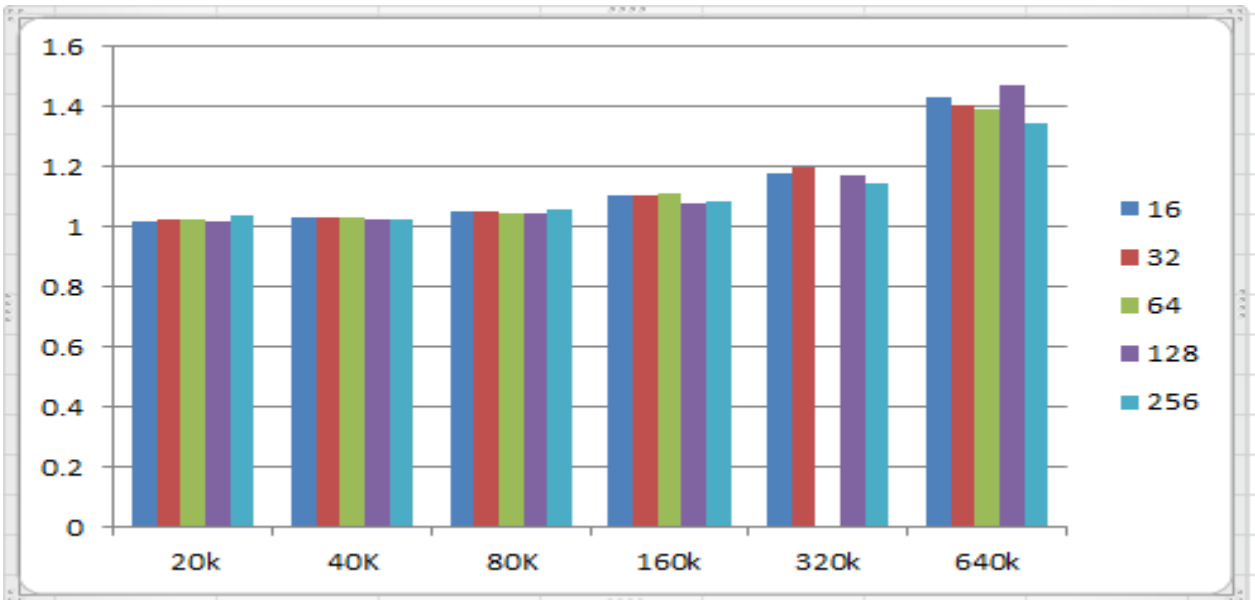
```
kukki@kukki-VirtualBox:~$ ./produce 320000 64
Initialization Time=0.000241994857788085937500000000000000000000000000000000000000000000 seconds
Transmission Time=1.194249868392944335937500000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 320000 64
Initialization Time=0.000216960906982421875000000000000000000000000000000000000000000000 seconds
Transmission Time=1.218366146087646484375000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./threads 320000 64
Initialization Time=0.000097990036010742187500000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000576019287109375000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 320000 64
Initialization Time=0.000019073486328125000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000041007995605468750000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./produce 320000 64
Initialization Time=0.000282049179077148437500000000000000000000000000000000000000000000 seconds
Transmission Time=1.185137033462524414062500000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 320000 64
Initialization Time=0.000284910202026367187500000000000000000000000000000000000000000000 seconds
Transmission Time=1.143244981765747070312500000000000000000000000000000000000000000000 seconds
child Process exited normally
```

				B		
		16	**32**	**64**	**128**	**256**
	20k	0.000111082	0.000101373	0.000148343	0.000175496	0.000158513
	40K	0.000468764	0.000191988	0.000158536	0.000105812	0.000109958
N	**80K**	0.000069023	0.000109241	0.00008177	0.000113128	0.000745627
	160k	0.000110441	0.000071809	0.000288009	0.000104718	0.000143662
	320k	0.000106383	0.000123425	0.000156217	0.000178339	0.000133786
	640k	0.000095422	0.000232363	0.0002260	0.000202226	0.0001705

Table1-Average Time Taken (in Seconds) for the Initialization of the Process

				B		
		16	**32**	**64**	**128**	**256**
	20k	1.0179042	1.023545406	1.02551521	1.02021111	1.0350722
	40K	1.029507	1.03343498	1.03199823	1.027214888	1.02452065
N	**80K**	1.051610782	1.051931441	1.044519565	1.046824422	1.054902811
	160k	1.102562	1.108210464	1.111191039	1.07649215	1.083130588
	320k	1.180836893	1.198034415	0.000155317	1.170143982	1.142547201
	640k	1.429518426	1.405445618	1.392810191	1.472415627	1.343374847

Table 2- Average Time Taken (in Seconds) for Transmitting data during the Process



We can calculate the mean deviation as well of the both Initialization and transmission of the data during the multi-processes as well.

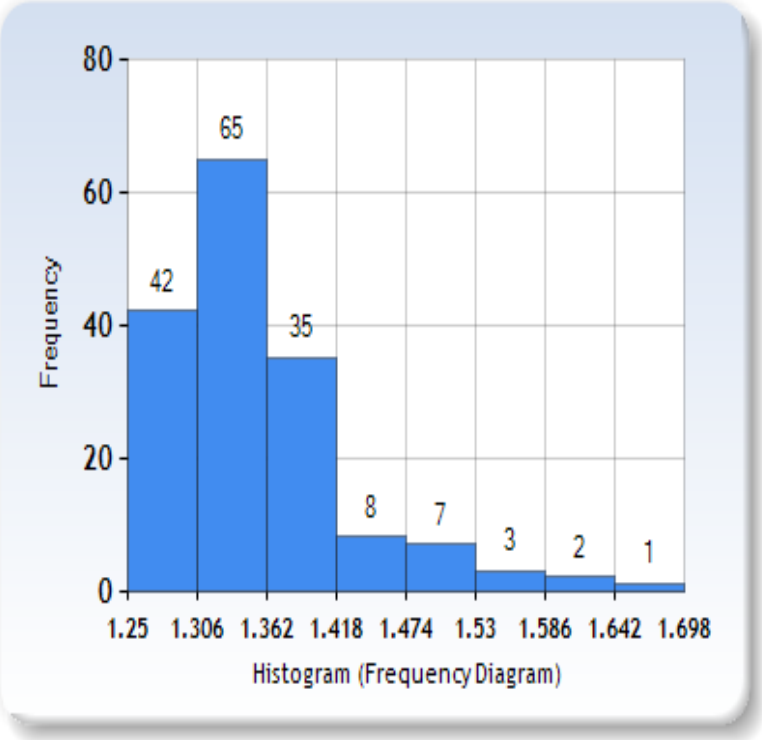
				B		
		16	**32**	**64**	**128**	**256**
	20k	0.00730991	0.006795333	0.001917461	0.009141629	0.008880186
	40K	0.009987555	0.005748728	0.008972166	0.011188329	0.008275983
N	**80K**	0.009079289	0.003544587	0.006660131	0.006926496	0.01262608
	160k	0.019456474	0.03789989	0.029724332	0.010634456	0.012745779
	320k	0.012232347	0.053931114	0.055510858	0.024921187	0.027530249
	640k	0.01677464	0.0418317	0.057497714	0.0732317	0.071534222

Table-4 for the Standard Deviation in the Transmission Time (Taken in Seconds)

Transmission Time for N=640000 and B=128

Frequency Table	
Class	*Count*
1.25-1.305	42
1.306-1.361	65
1.362-1.417	35
1.418-1.473	8
1.474-1.529	7
1.53-1.585	3
1.586-1.641	2
1.642-1.697	1

Your Histogram	
Lowest Score	1.25
Highest Score	1.67
Total Number of Scores	163
Number of Distinct Scores	29
Lowest Class Value	1.25
Highest Class Value	1.697
Number of Classes	8
Class Range	0.056



Similarly with the Multithreads, we have Average Time for the Time of Initialization of the System can be calculated as:

MULTI-THREAD IMPLEMEMTATION

```
kukki@kukki-VirtualBox:~$ ./threads 640000 128
Initialization Time=0.000077962875366210937500000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000722169876098632812500000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 640000 128
Initialization Time=0.000080108642578125000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000996828079223632812500000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 640000 128
Initialization Time=0.000019788742065429687500000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000031232833862304687500000000000000000000000000000000000000000000 seconds
^[[Akukki@kukki-VirtualBox:~$ ./threads 640000 128
Initialization Time=0.000072002410888671875000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.001386165618896484375000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 640000 128
Initialization Time=0.000022172927856445312500000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000033855438232421875000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 640000 128
Initialization Time=0.000076055526733398437500000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.002051115036010742187500000000000000000000000000000000000000000000 seconds
```

Multi-Threads is the ability of the Operating System in a single core or in a Multi-Core processor to execute the multiple-processes concurrently. As in a sharp contrast to the Multi-Processes, multithreading aims to increase the utilization of a single core by using thread-level as well as instruction level parallelism.

				B		
		16	**32**	**64**	**128**	**256**
	20K	0.000057536	0.000068182	0.000043362	0.000094174	0.000022952
	40K	0.000041624	0.00007258	0.000074156	0.00004856	0.000048254
N	**80K**	0.000563403	0.000066606	0.00006216	0.00003478	0.000069126
	160K	0.000220854	0.000072125	0.000059893	6.21713E-05	0.00004678
	320K	0.000392129	6.78643E-05	0.000061023	4.77757E-05	5.79527E-05
	640K	0.00005162	0.00003258	0.000060178	0.000049822	0.000035622

Table-5 Average time (in the Seconds) for the System Initialization

				B		
		16	**32**	**64**	**128**	**256**
	20K	0.000093611	0.00061998	0.001245473	0.000870562	0.0000378
	40K	0.000343977	0.00072323	0.002283172	0.002923172	0.0003514
N	**80K**	0.000520358	0.00044436	0.00038938	0.00042356	0.00061758
	160K	0.000326208	0.0005925	0.001305975	0.001405765	0.000335593
	320K	0.000450121	0.00051842	0.000847677	0.000914762	0.000476587
	640K	0.000848613	0.00020921	0.000611999	0.00043014	0.000414746

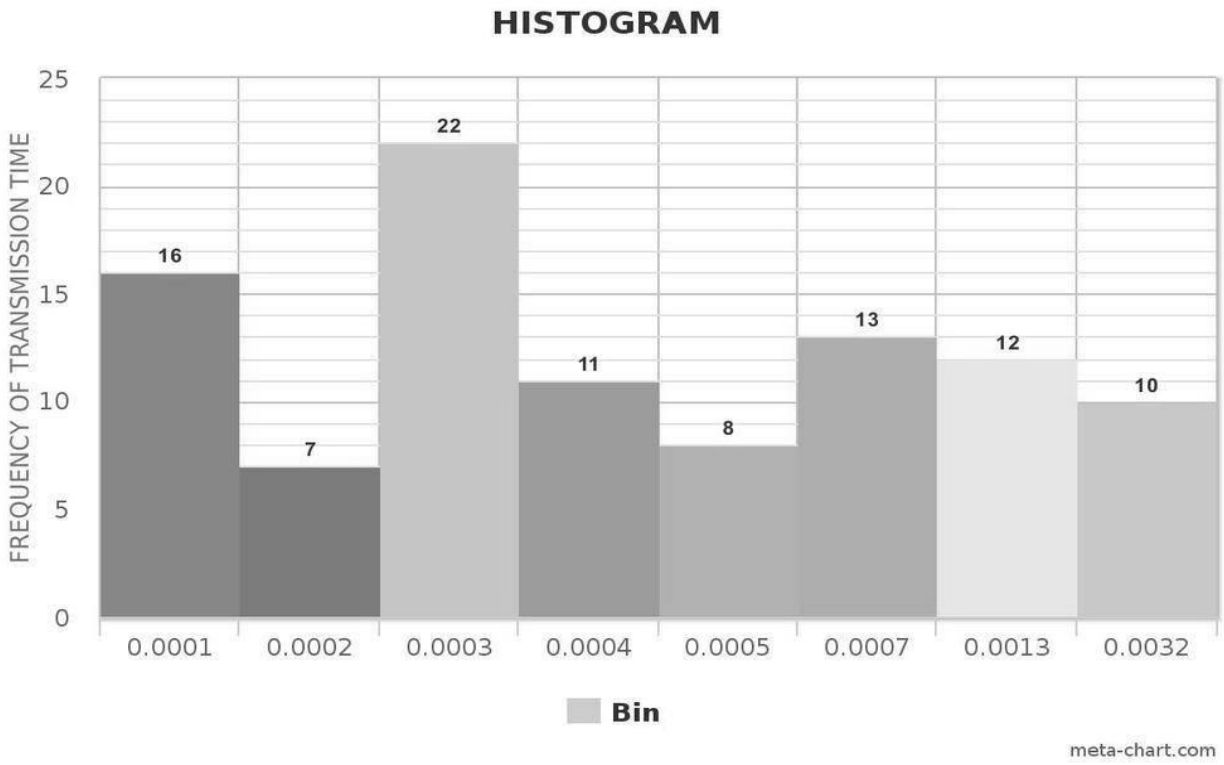
Table-6:- for the Average time (in the Sec.) for transmission of Data.

Similarly in this case, we can calculate the Standard Deviation in the Average of time obtained both in the Time taken to initialize as well time taken by the System to transmit the Data.

				B		
		16	**32**	**64**	**128**	**256**
	20K	4.3876E-05	0.000574851	0.001661443	0.001340481	8.97524E-06
	40K	0.000639473	0.000729426	0.002248449	0.005728895	0.000660191
N	**80K**	0.000724308	0.000355038	0.000509903	0.000754491	0.001183531
	160K	0.000656798	0.001760047	0.000304882	0.001103468	9.86801E-05
	320K	0.000443833	0.001719197	0.001419501	0.000749935	0.002034426
	640K	0.001101884	0.000358595	0.000738111	0.000805409	0.000764435

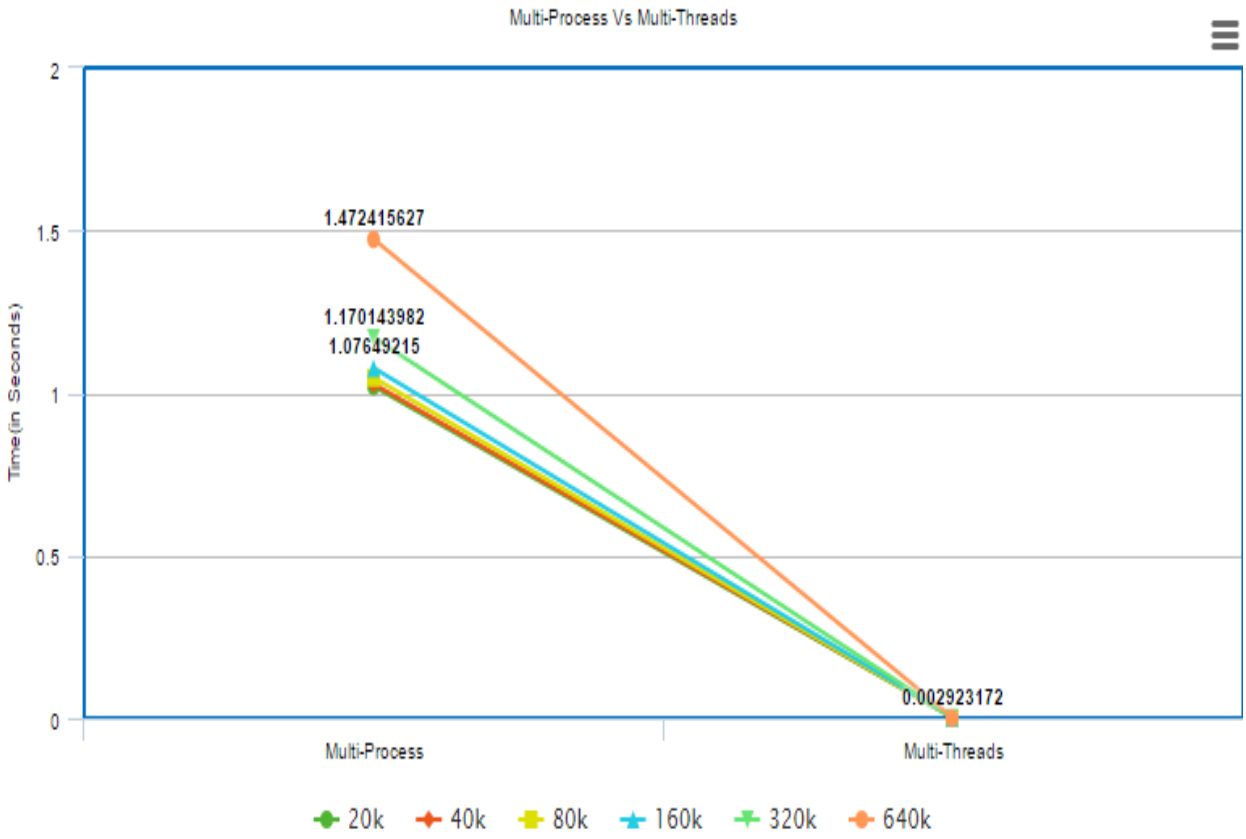
Table-8:- Standard Deviation of Time (in Sec.) for the Transmission in Multi-Threads.

Now we can plot the Histogram of the frequencies and the values and we would see that the value 0.003 Seconds is the most common one appeared during the Execution.



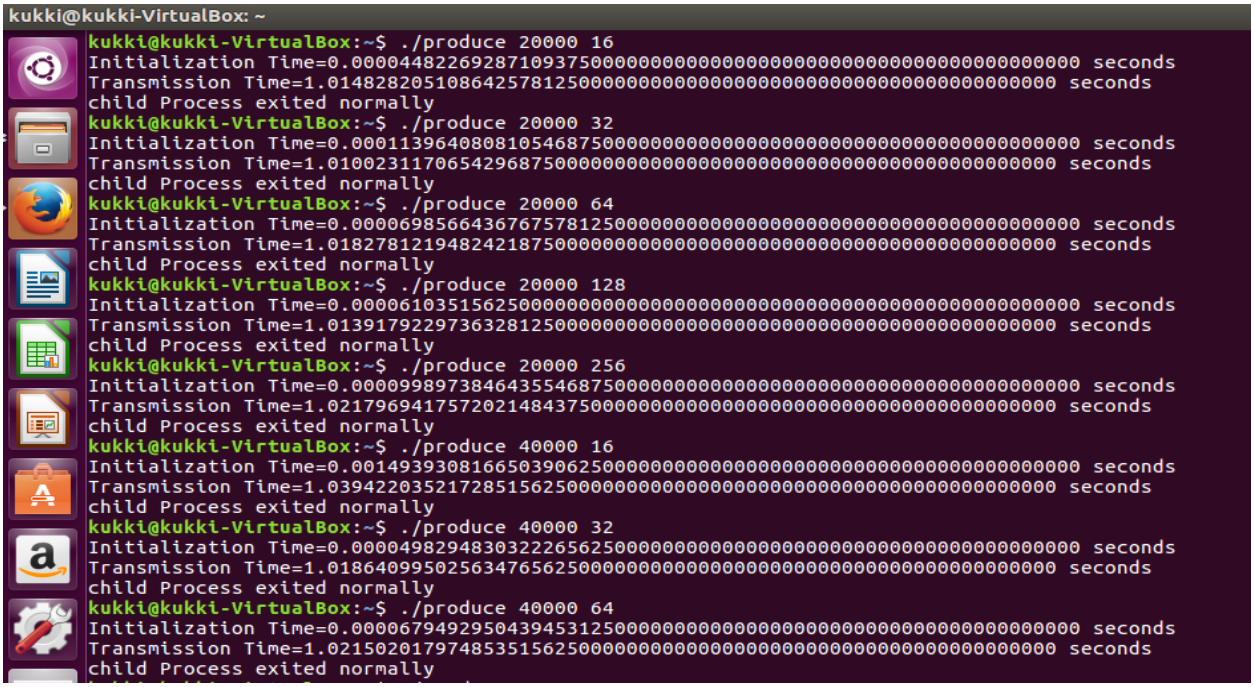
Comparison Between the two Approaches:-

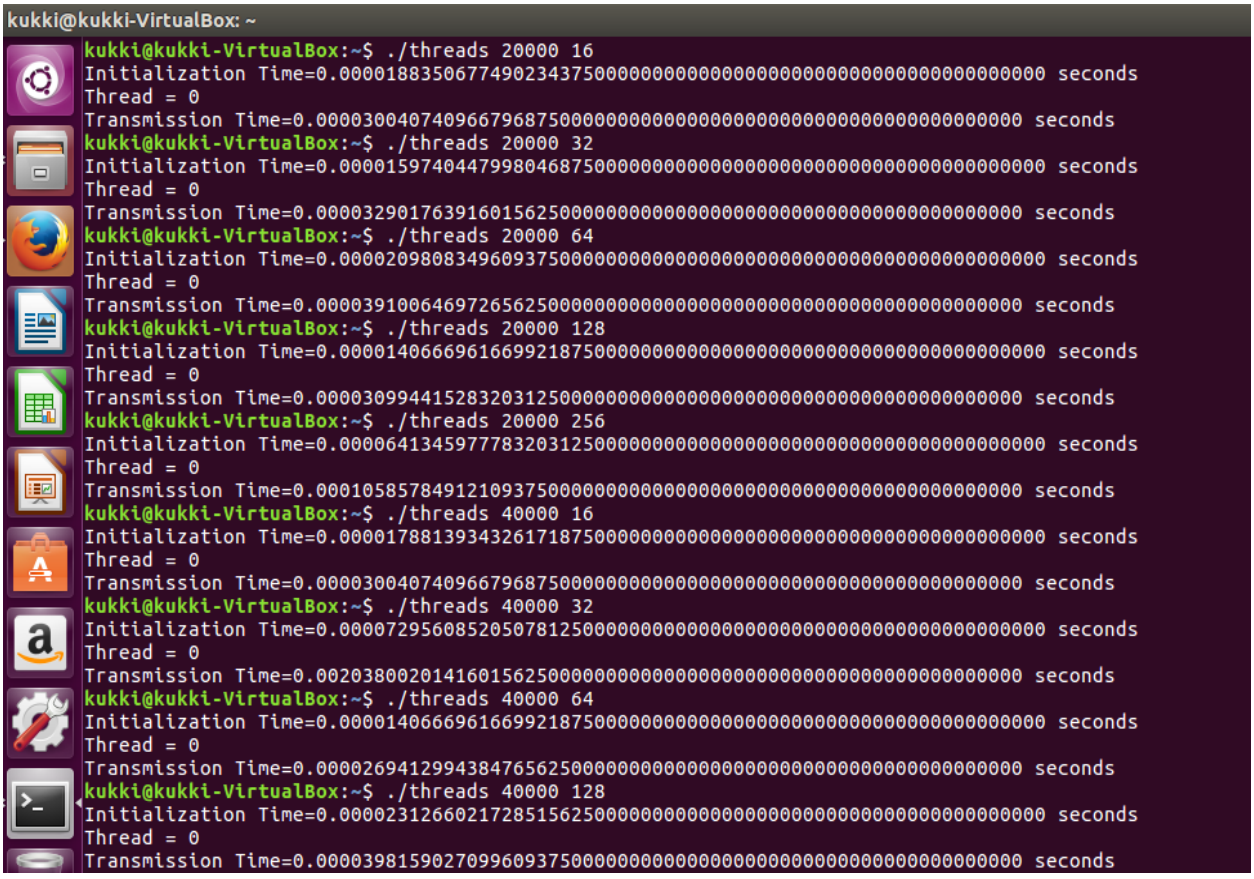
Timing Data for N=640000 and B=128			
Approach	Average Initial Time	Transmission Time	Frequency Distribution
Multi-Process	0.000202226	1.472415627	0.0732317
Multi-Thread	0.000049822	0.00043014	0.000805409



As Evident we can see that the Average Transmission time of the Multi-Thread Approach is way too much faster than the multi-Process Approach. Thus clearly, Multi-threaded implementation outperforms Multi-Process in terms of the Implementation and Speed.
In the Particular case where N=640000 and B=128, we can see that the Multithreaded approach is 300 times faster than the Multi-Process approach. The one reason for this high speed implementation could be that the threads are light weight as compared to the Processes and hence it is easy for the threads to switch in between as Compared to the Processes.

Another reason which could contribute to the lower speed of the Multi-Process is the fixed size of the Message Queue involved in the Multi-Processes where as in the Multi-Threads the Buffer uses has greater size and hence it is easy for the threads to fill up and empty the Queue.


kukki@kukki-VirtualBox: ~
kukki@kukki-VirtualBox:~$ ./produce 20000 16
Initialization Time=0.000044822692871093750000000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.014828205108642578125000000000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 20000 32
Initialization Time=0.000113964080810546875000000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.010023117065429687500000000000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 20000 64
Initialization Time=0.000069856643676757812500000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.018278121948242187500000000000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 20000 128
Initialization Time=0.000061035156250000000000000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.013917922973632812500000000000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 20000 256
Initialization Time=0.000099897384643554687500000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.021796941757202148437500000000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 40000 16
Initialization Time=0.001493930816650390625000000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.039422035217285156250000000000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 40000 32
Initialization Time=0.000049829483032226562500000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.018640995025634765625000000000000000000000000000000000000000000000000000 seconds
child Process exited normally
kukki@kukki-VirtualBox:~$ ./produce 40000 64
Initialization Time=0.000067949295043945312500000000000000000000000000000000000000000000000000 seconds
Transmission Time=1.021502017974853515625000000000000000000000000000000000000000000000000000 seconds
child Process exited normally


kukki@kukki-VirtualBox: ~
kukki@kukki-VirtualBox:~$ ./threads 20000 16
Initialization Time=0.000018835067749023437500000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000030040740966796875000000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 20000 32
Initialization Time=0.000015974044799804687500000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000032901763916015625000000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 20000 64
Initialization Time=0.000020980834960937500000000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000039100646972656250000000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 20000 128
Initialization Time=0.000014066696166992187500000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000030994415283203125000000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 20000 256
Initialization Time=0.000064134597778320312500000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000105857849121093750000000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 40000 16
Initialization Time=0.000017881393432617187500000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000030040740966796875000000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 40000 32
Initialization Time=0.000072956085205078125000000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.002038002014160156250000000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 40000 64
Initialization Time=0.000014066696166992187500000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000026941299438476562500000000000000000000000000000000000000000000000000 seconds
kukki@kukki-VirtualBox:~$ ./threads 40000 128
Initialization Time=0.000023126602172851562500000000000000000000000000000000000000000000000000 seconds
Thread = 0
Transmission Time=0.000039815902709960937500000000000000000000000000000000000000000000000000 seconds

Cost Comparison between Multi-threading and Multi-processing:-
The timing analysis of the both of the approaches clearly gives rapport to the Multi-threading approach as Producer-Consumer implementation is significantly faster than the process implementation.
In the case of the Multi-threading, Data is shared amongst the other various threads while the process is running and thus this implementation is quite useful in the tasks which would require large amount of data to be shared among itself. More over the Threads always use less resource than the process. However, being too fast to execute threads does have certain disadvantages. First of all, during the multi-threading with in the same program, all the threads have to run or undergo the same execution. More over there is flow of errors in the case of the multithreading. If there is an error in one thread that causes the data to corrupt then that error would affect all the other threads as well.

More over during the execution as well the implementation of the program, I realized that the debugging of the program employing multi-threading is significantly harder as compared to the Multi-processing. More over threads do needs assistance from the other application such as the mutex and Semaphore which makes the whole program to be way more complex as well
The multiprocessing always has an advantage that the fault doesn't propagate into failure. Hence fault arising due to the multi-threading is always a localized one and doesn't bother or affect the neighbouring processes which are going on side by side.
As evidently cleared from the timing consideration, we could infer that though the Multi-processing is an expensive process which is significantly slower than the Multi-threading but it is more reliable and less susceptible to the race conditions and deadlocks.
So we could conclude that though the Multi-threading is way more faster than the Multi-processing but this speed comes at the cost of the Deadlocks and Race Conditions. Similarly on the other hand, Multi-Processing is way more reliable and significantly not at al prone to the deadlocks and Race conditions but reliability comes at the cost of speed.

Note:-In this given assignment, we have been given the inbuilt hard value of the size of the message queue is 10 which has to be changed to 256(maximum value for the Buffer). And hence to calibrate the value of the Buffer it has to be changed using

$ sudo echo 256>/proc/sys/fs/mqueue/msg_max and then manually changing the values to 256 in the Ubuntu.