Lab 3 Report

by

Hejia Wang,

Xiaoyuan Lin

ECE 254

November 20, 2018

- **Table:**

N	B	P	C	AVG Time Processes	AVG Time Threads	STD Time Processes	STD Time Threads
100	4	1	1	0.981176	0.39553	0.05897709	0.0754345385086
100	4	1	2	1.012454	0.38239	0.02099828	0.0596084046087
100	4	1	3	0.87131	0.421372	0.21348439	0.115214398475
100	4	2	1	0.734958	0.377542	0.04052743	0.0611043389294
100	4	3	1	0.727502	0.38928	0.06413731	0.0765559507811
100	4	2	2	0.590026	0.359064	0.0298467	0.0539694349053
100	4	3	3	0.601136	0.382042	0.02866171	0.0870381768881
100	8	1	1	0.675418	0.362358	0.06247949	0.0767645350145
100	8	1	2	0.57313	0.370266	0.0562337	0.0587200753065
100	8	1	3	0.570566	0.403288	0.02589142	0.087074204387
100	8	2	1	0.718438	0.359326	0.04722108	0.0639773688424
100	8	3	1	0.71075	0.38188	0.06881122	0.0762803880431
100	8	2	2	0.577486	0.36221	0.02852048	0.08477031261
100	8	3	3	0.602288	0.366714	0.03121758	0.0908226414723
398	8	1	1	0.832768	0.712496	0.08592973	0.0391117115964
398	8	1	2	0.94187	0.728638	0.05584429	0.0544376244522
398	8	1	3	1.008484	0.768954	0.0732217	0.033453368799
398	8	2	1	0.969796	0.640708	0.05969233	0.0639302646326
398	8	3	1	1.017752	0.710864	0.06531354	0.0419312950432
398	8	2	2	0.790532	0.502364	0.08254677	0.0538023745201
398	8	3	3	0.923186	0.51548	0.11977469	0.0670289907428

- **average time and standard deviation**

  Since our program was run on a real Linux system, there would be many external factors that could affect the final result and datum. Not only which server we chose to use, how crowd that server was when we run our program and many other factors could affect the accuracy of the result, but also how tasks were scheduled by the operating system and how the OS decided to switch among our tasks would also affect the result. None of these factors we could control. Thus, we are using average time to minimize the variance in our results and obtain a more accurate report. Also, we are using standard deviation to check if our datum is tight enough (accurate, few extreme outliers) and reasonable.

  Although we use average time and standard deviation to minimize variances, there are still some in our final datum. However, the general conclusion should still be valid.

- **Processes VS. Threads**

  According to Figure 1, 2 and 3, we could notice that the overall performance of Process and Thread. As we can see, thread has a much better performance than process for different parameters.

  The reason is that threads run in shared memory space, while processes use a POSIX message queue to communicate. Using POSIX message queue would require processes to have an additional step and use external API for datum proceeding. Even though we use semaphore and mutex in thread code, which would slow down the speed, the shared memory space (of thread) is still much more direct and fast to access comparing to the inter-process communication.

  Another point is: Creating a new thread is much faster than creating a new process (using fork()), thus when coming to the multi-consumer condition, threads keep to have a lower average time comparing to processes, which enable threads always finish faster.

  Although processes are slower, they also have their own advantages: processes are safer than threads. The reason is processes are independent from each other, if one process got stuck or hit an error, it would not affect other processes. However, if one error occurs in a process, it would crash the whole process as well as all other threads.

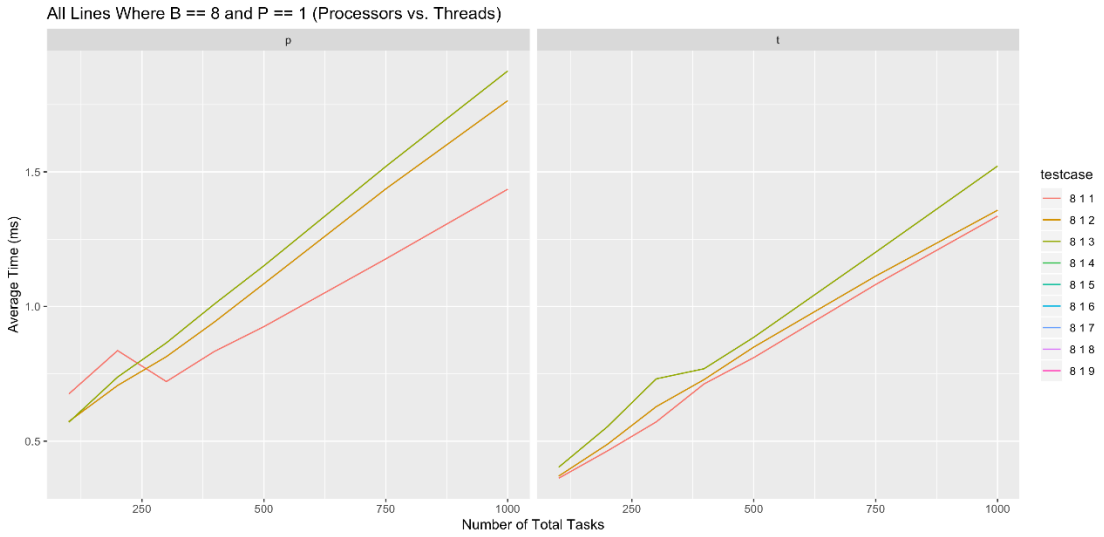

Figure 1: Comparing process and thread in single producer (1)

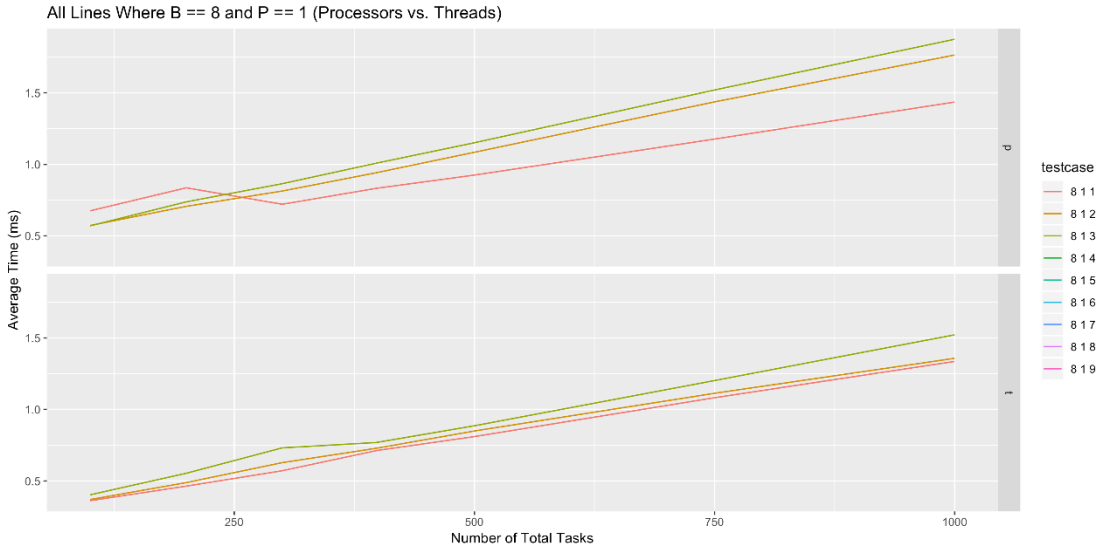

Figure 2: Comparing process and thread in single producer (2)

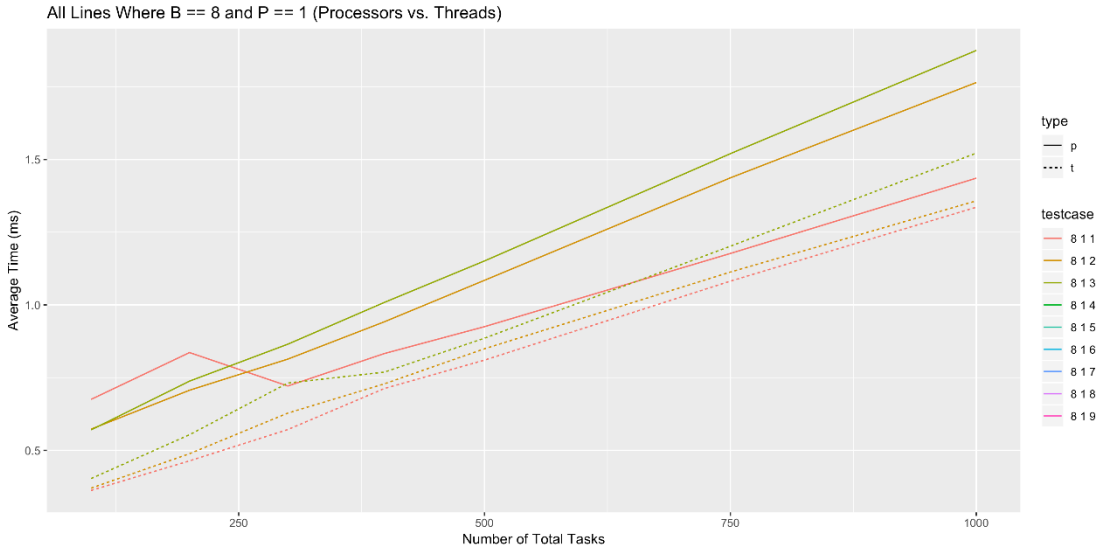

Figure 3: Comparing process and thread in single producer (3)

Change in variables

1) **Change in number of items:**
   According to Figure 4: Increasing the numbers to be generated by producer would increase the amount of work to be done. Thus, it increases the average time to complete the tests.
2) **Change in buffer size:**
   According to Figure 4: Buffer size has little impact on the performance, expect for specific edge cases. The reason for this is that, both producers and consumers runs very quickly, as a result, the speed of data flow would not be restricted by the size of the buffer: buffer is always been putting in and taking out datum, rarely becomes full or empty.
3) **Change in number of Producers**
   According to Figure 4: Overall, increasing the number of producers would make the performance worse. This is because producers are fast enough themselves for this simple generating numbers task. As a result, the benefit (the total time saved) of creating new producers and share the simple task among multiple producers is not even huger than the drawback (the total time waste) for creating these producers. However, there is one case that the performance would be better, we would discuss this later.
4) **Change in number of Consumers**
   According to Figure 4: Increasing the number of consumers would also lead to a worse performance for processes. However, when the number of producers and the number of consumers are close, the performance would become better (the smaller the number of producers/consumers when the numbers are close, the better performance). The reason why increasing consumer would lead to a worse performance is the same as that in (3) when we discuss relationship between performance and number of producers. The reason why when the numbers are close, the performance would be better is that: Since producers have a similar average run time comparing to consumers, when the numbers of producers and consumers are close, neither of them have to wait for the other and be blocked when running the program.

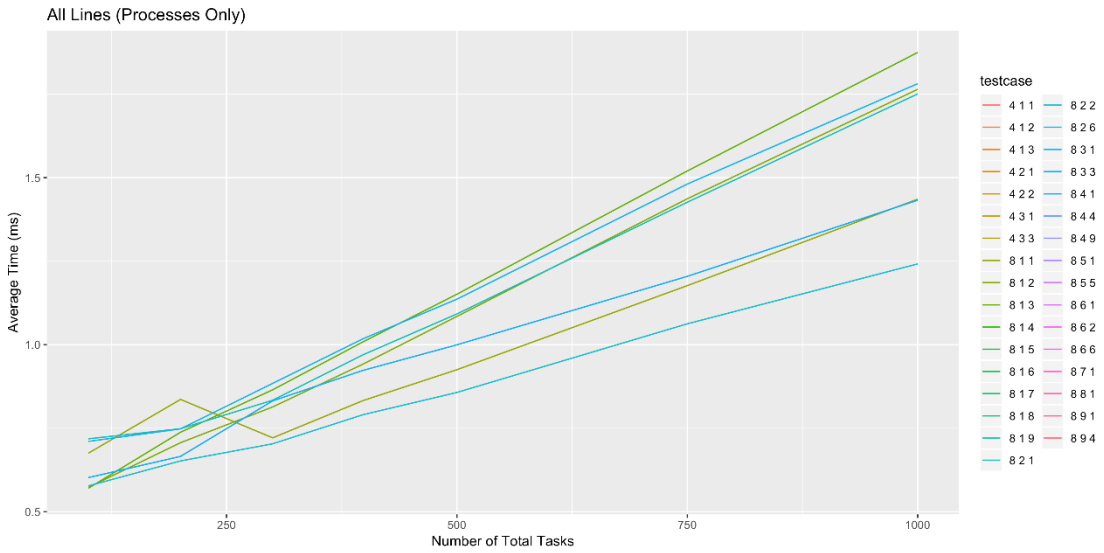

Figure 4: Process performance under different parameters