Pawel Derkacz – pd304
Deep Patel – dp767

Input1: <AaAa aaaaaaa AA98515A4A7A33A8A567A A -=/' A aaaa AAA aa A a> * 10,000 + new line characters. Total of 600,000 bytes, exactly.

Tests for threads and multi-processes done by use of the *time* command in terminal. For multi-processes, time only needs to be counted on the parent, since it waits for its children.

Number of Parts	Time taken by threads	Time Taken by multi-processes	Number of Parts	Time taken by threads	Time taken by multi-processes
1	.02s	.031s	11	.01s	.009
2	.028s	.029s	12	.007s	.011
3	.011s	.015	13	.011s	.014
4	.016s	.019	14	.018s	.015
5	.024s	.011	15	.019s	.014
6	.015s	.009	16	.017s	.021
7	.021s	.014	17	.022s	.011
8	.008s	.01	18	.016s	.02
9	.024s	.024	19	.01s	.007
10	.011s	.016	20	.01s	.016

Average time for threads in first half: 0.0178 seconds
Average time for threads in second half: 0.014 seconds
Average time of threads: 0.0159 seconds
Avg. bytes/second (floored) = 37735849 bytes/second (≈ 36 MB/sec)

Average time for multi-processes in first half: 0.0178 seconds
Average time for multi-processes in second half: 0.0138 seconds
Average time for multi-processes: 0.0158 seconds
Avg. bytes/second (floored) = 37974683 bytes/second (≈ 36.2 MB/sec)

Conclusion:

It seems that both forms take about the same amount of time, and vary with the amount of data usage. Though we didn't test for it, multi-processing takes up more space than multi-threading. Therefore, for this type of project, multi-threading seems to be the way to go. There are instances where multi-processing must be used, but if the project allows for multi-threading, then that is the better bet, unless there is truly a LOT of data, then multi-processing may pay off with the amount of data used vs. speed (since it is that tiniest bit faster, makes a difference over large amounts).