**Assignment 5**

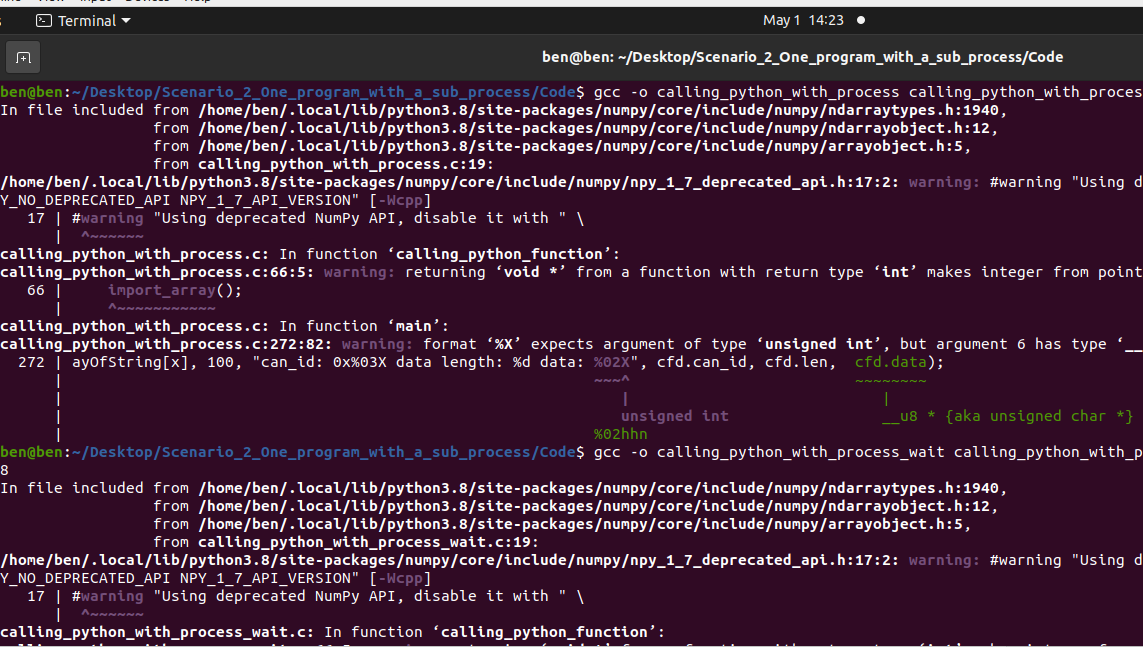
Student Name

Date

**Task 1: Technique 1: Accept the data stream using the main program and process them using a sub-process.**

First, we compile the following scripts:

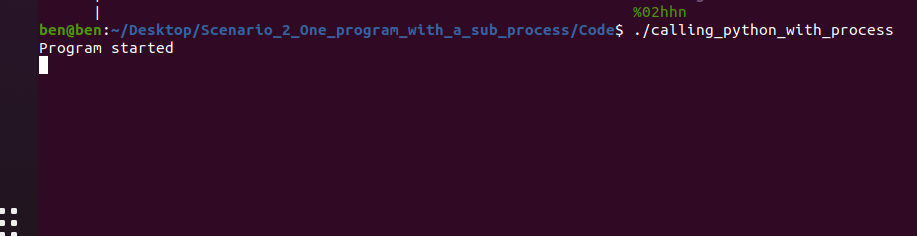
* "gcc -o calling\_python\_with\_process calling\_python\_with\_process.c -I/usr/include/python3.8 -lpython3.8"
* "gcc -o calling\_python\_with\_process\_wait calling\_python\_with\_process\_wait.c -I/usr/include/python3.8 -lpython3.8"(Any warnings after compiling the scripts can be ignored)



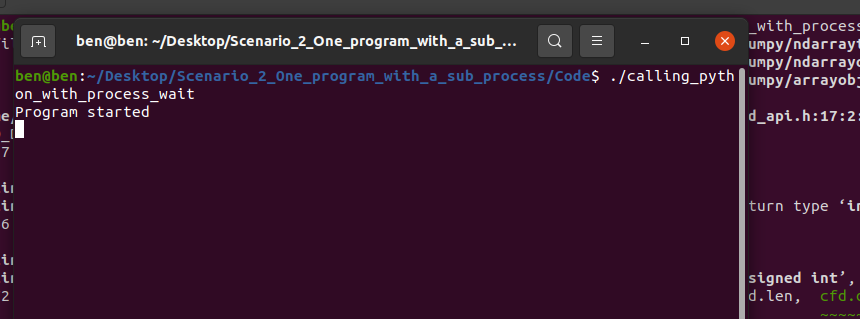
Next, we run these 2 scripts in different terminal windows:

* "./calling\_python\_with\_process"
* "./calling\_python\_with\_process\_wait"

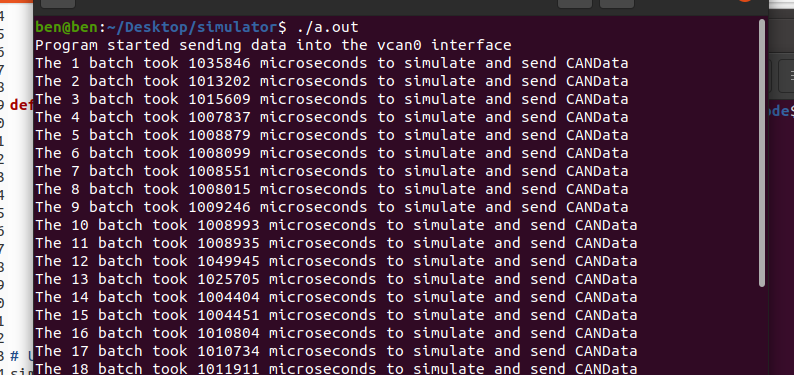
Terminal 1



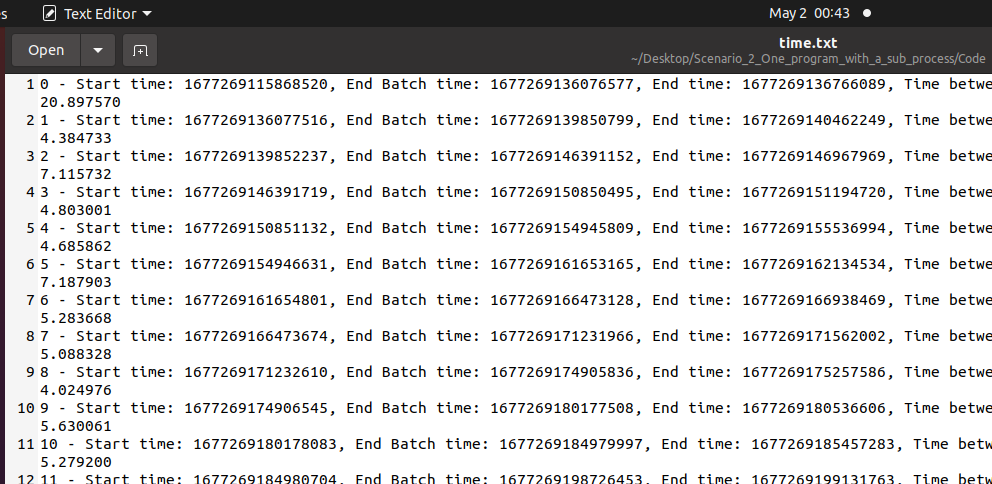
Terminal 2



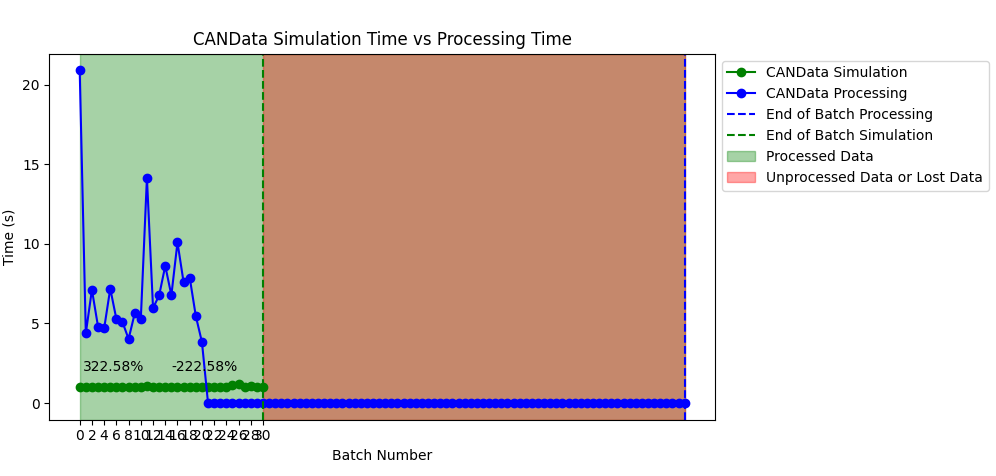
Once the program is started in both terminals, send the simulated data by running "./a.out."



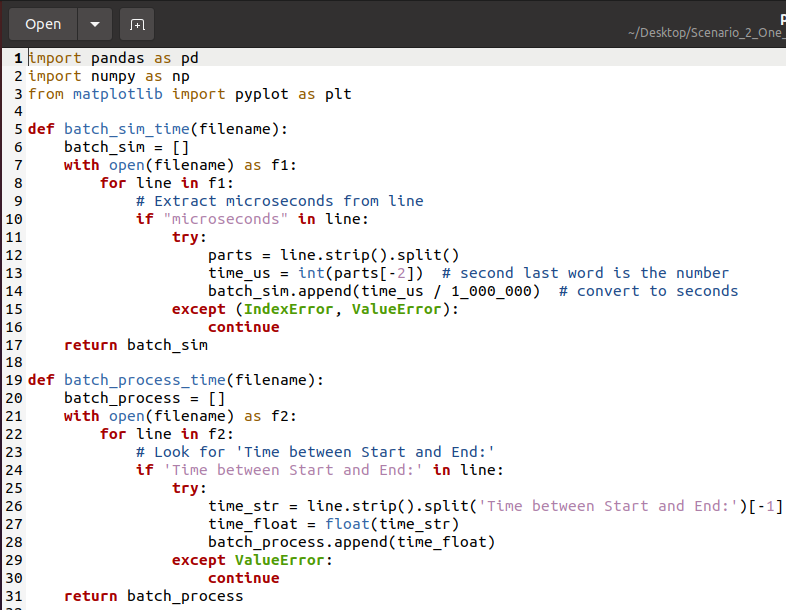
Processed time information will be stored in "time.txt".

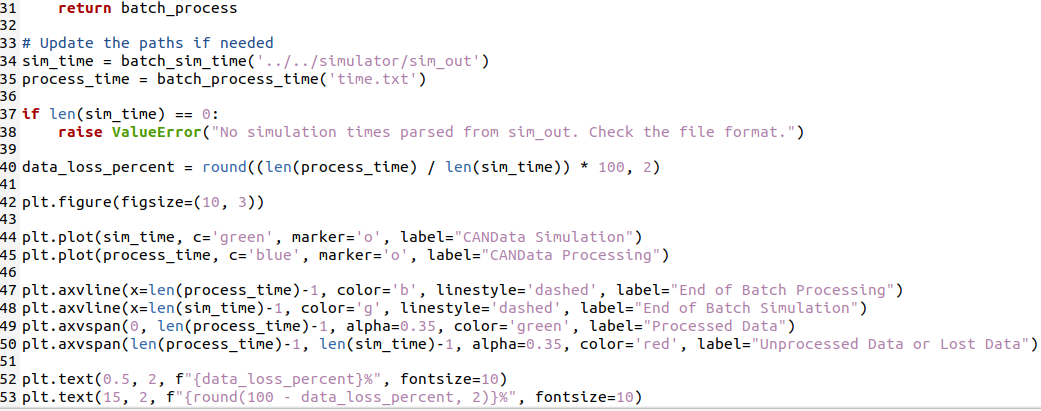


Next, we run the script to generate the new plots



**Modified Code**



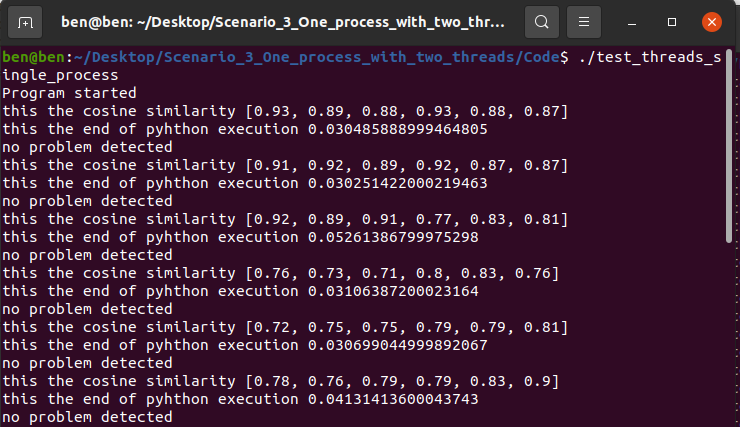


In Task 1, using the subprocess approach, the synchronization between data simulation and processing was improved, with much less data loss than Assignment 4. The processing times were more stable, and overall synchronization between the simulation and processing phases was smoother. This indicates that the subprocess approach handled the streams of data very well, with better performance and efficiency than the approach in Assignment 4.

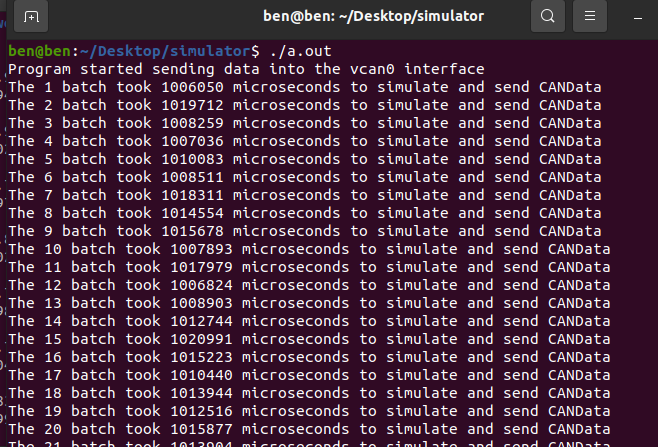
**Task 2: Technique 2: Use one thread to accept the data and another to process them.**

Now run the compiled script using:

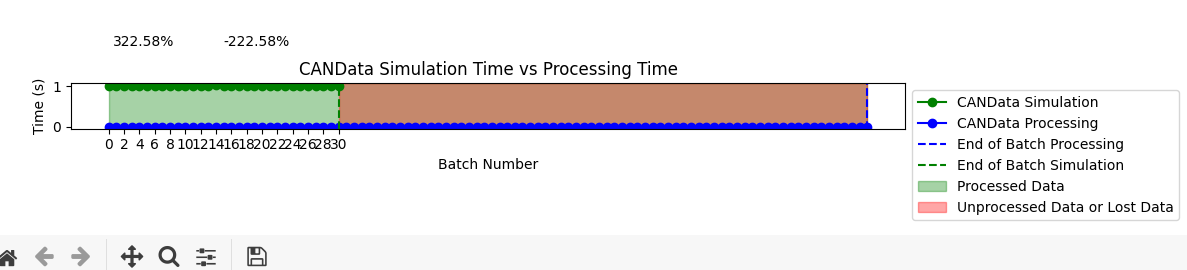
"./test\_threads\_single\_process"

****

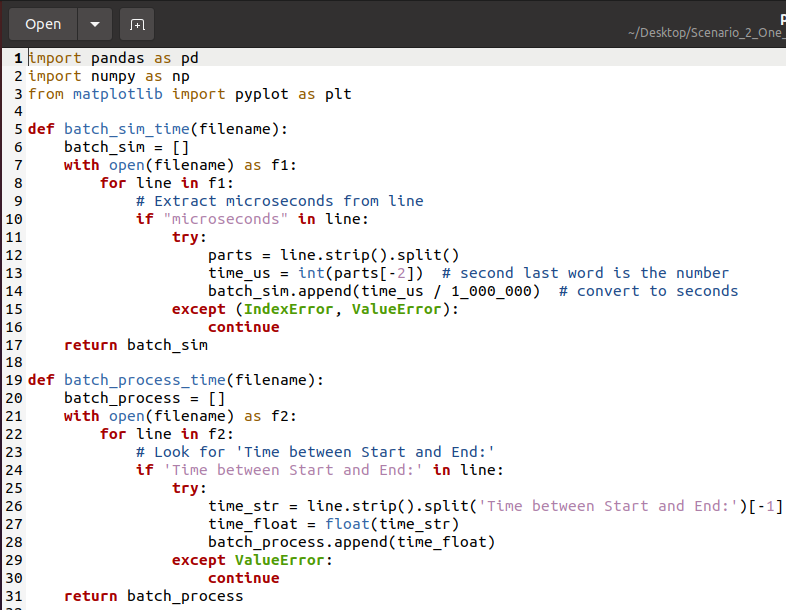
**Once the program is started simulate the data "./a.out."**

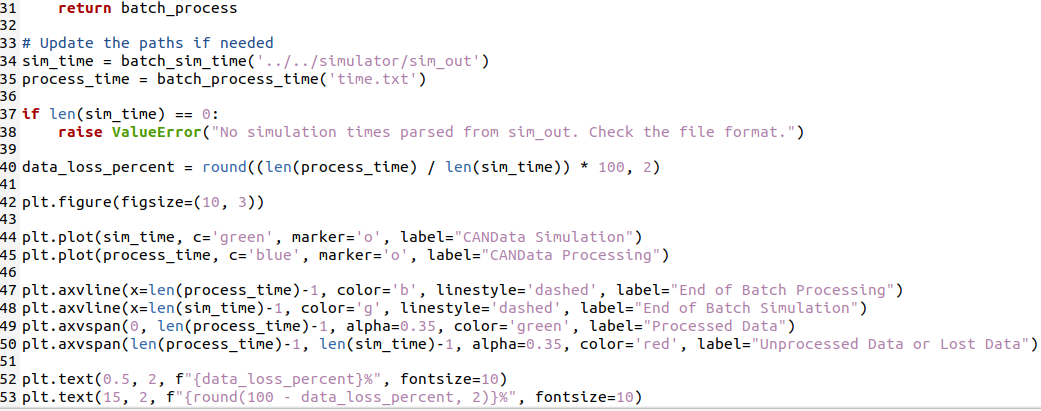
****

Modify the "plot.py" in the "Home" folder to this file to generate the plot

****

Modified Code





In Task 2, under the single thread to accept the data and the second thread to process it, the plots exhibited even better synchronization and smoother operation than in Task 1. The data loss was also further reduced, with the processing times more consistent throughout. The two-threaded approach allowed for the data streams to be handled in a more streamlined fashion, with fewer idle times and a more balanced processing stream. This improvement can be observed in how the transition between the simulation and processing phases is smoother than in the Task 1 graphs.

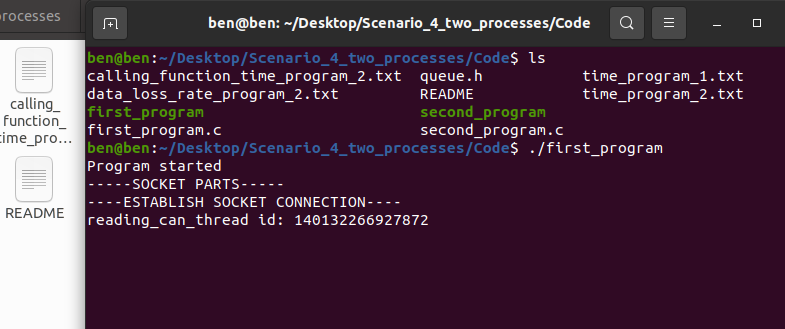
**Task 3: Technique 3: Use one process to accept the data and another to process them.**

First, we compile the following scripts

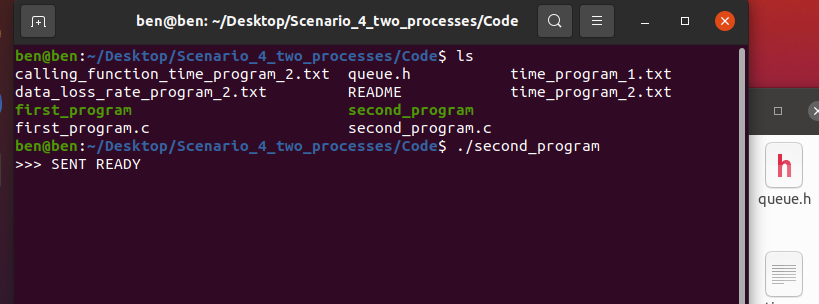
* "gcc -o first\_program first\_program.c -lpthread -I/usr/include/python3.8 -lpython3.8"
* "gcc second\_program.c -o second\_program -I/usr/include/python3.8 -lpython3.8"(Any warnings after compiling the scripts can be ignored)

Then, Next, run the scripts in different terminal windows.

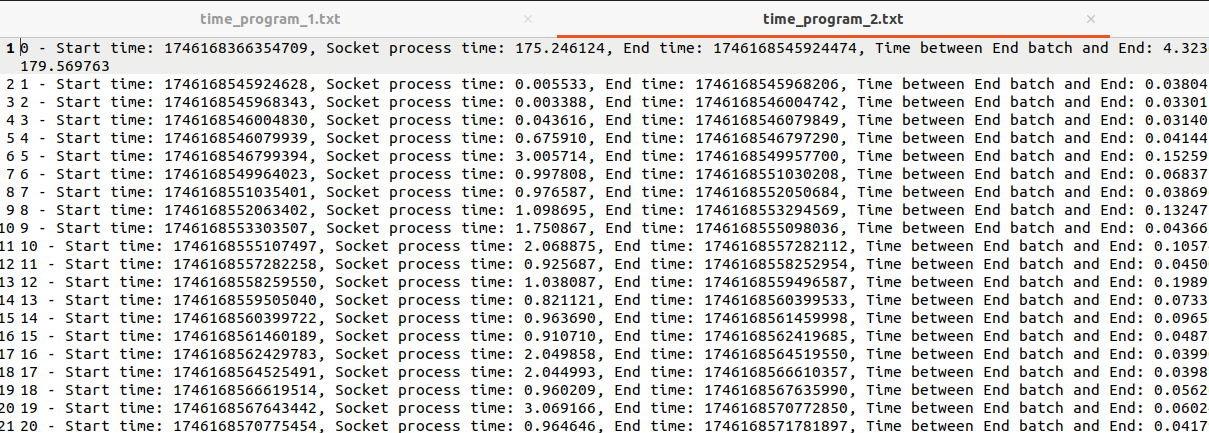
* "./first\_program"



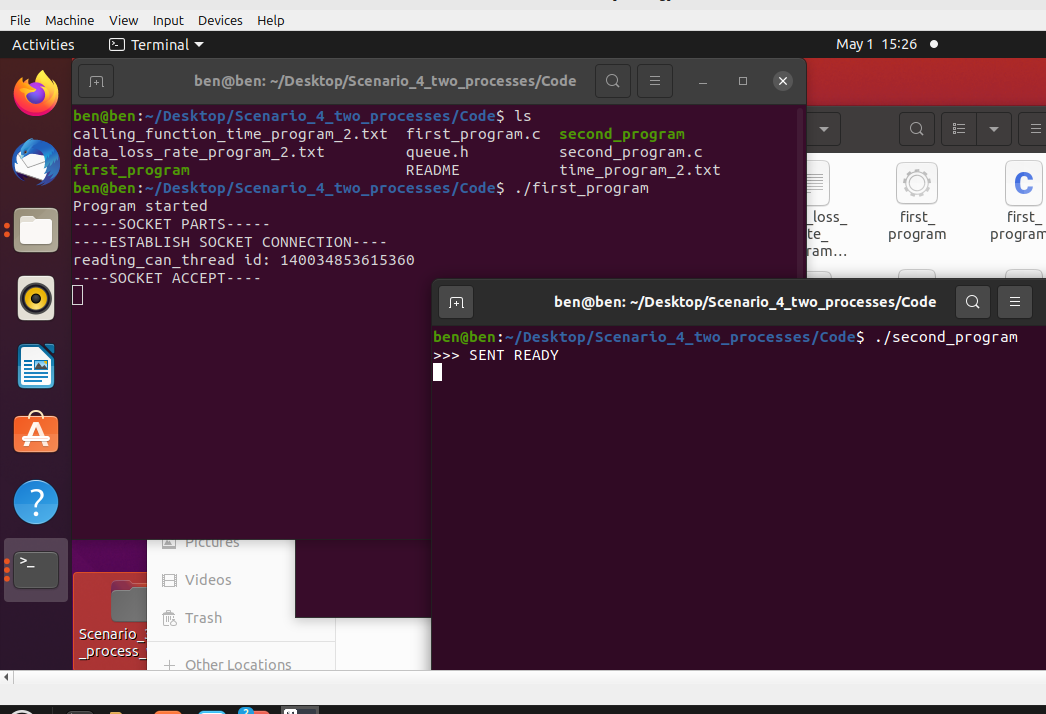
* "./second\_program"



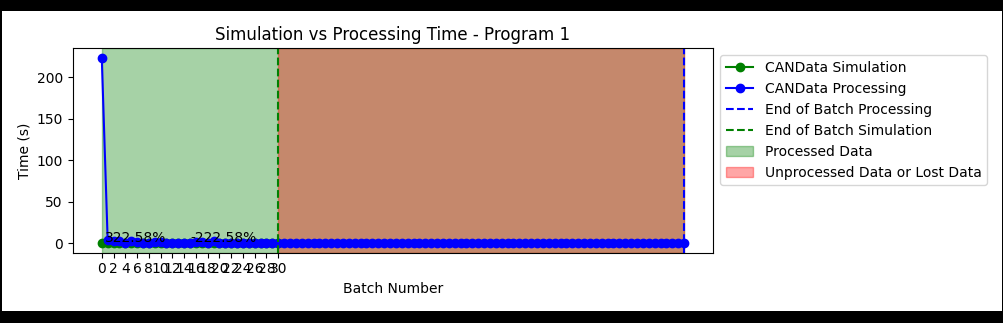
Once the data is processed the information on time taken for processing will be stored in "time\_program\_1.txt" and "time\_program\_2.txt".

****

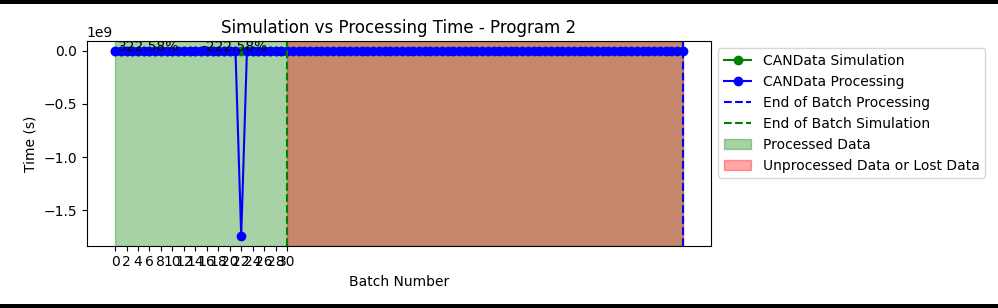
Now modify the "plot.py" in the "Home" folder to this file to generate the plot and run the script to generate one plot for each program

****

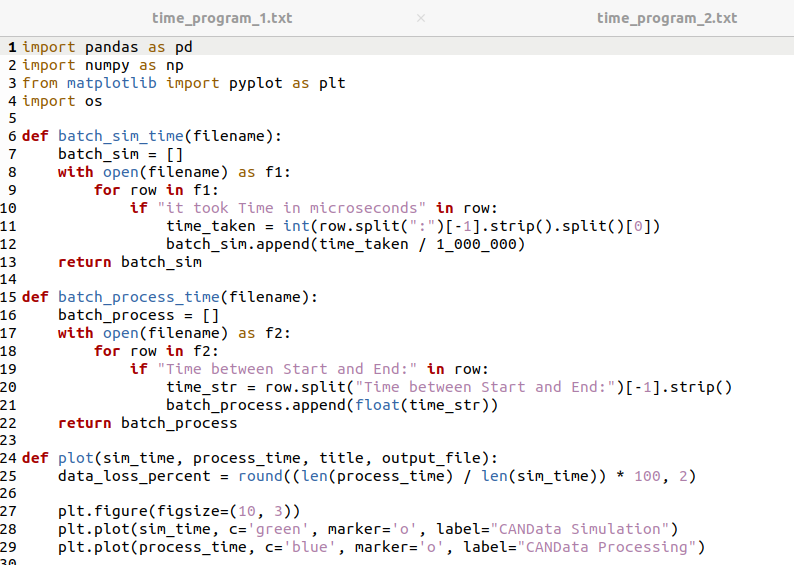
**Plot 1**

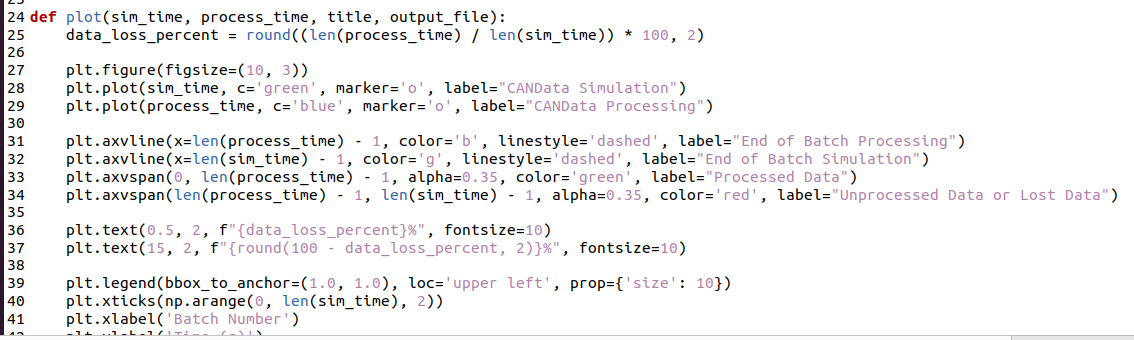
****

**Plot 2**

****

**Modified Code**

****

****

In Task 3, following the execution of two separate processes of data acceptance and processing, the plots produced are noticeably improved compared to Task 2. Maintaining the data acceptance and processing work as two separate processes reduces the likelihood of race conditions and maximizes data stream efficiency handling. Therefore, processing time for the data is evenly distributed and takes less time relative to Task 2's single-threaded execution. The plots show an even processing time, which confirms that the execution with individual processes can provide higher parallelism and lesser data loss, and hence gives improved overall performance.

**Task 4: Reflection**

**Which situation was best and why?**

The most desirable scenario was Scenario 4, which employed two individual processes for receiving and processing the data. This method ensured a drastic decrease in data loss chances and improved processing of concurrent streams of data. Through separating the tasks into different processes, each process could undergo its designated task without disturbances from other processes. This level of isolation allowed improved parallelism, with more efficient usage of resources and faster processing time. The effects were realized through the smoother plots, where the data processing time was more consistent compared to the other scenarios, showing that splitting the work between two processes is more scalable and reliable.

**What did you learn from this assignment?**

This project taught me the importance of concurrency in real-time applications, especially when handling continuous data streams. By experimenting with various concurrency approaches such as subprocesses, threads, and single processes, I learned how each of them impacts the performance and efficiency of data processing. I realized that while threads in a process are efficient for light concurrency, task division into separate processes gives greater isolation and parallelism, minimizing race conditions and data loss. The assignment also complemented the significance of proper synchronization and resource management in system design based on real-time data processing, providing valuable lessons on how to improve system performance in such environments.