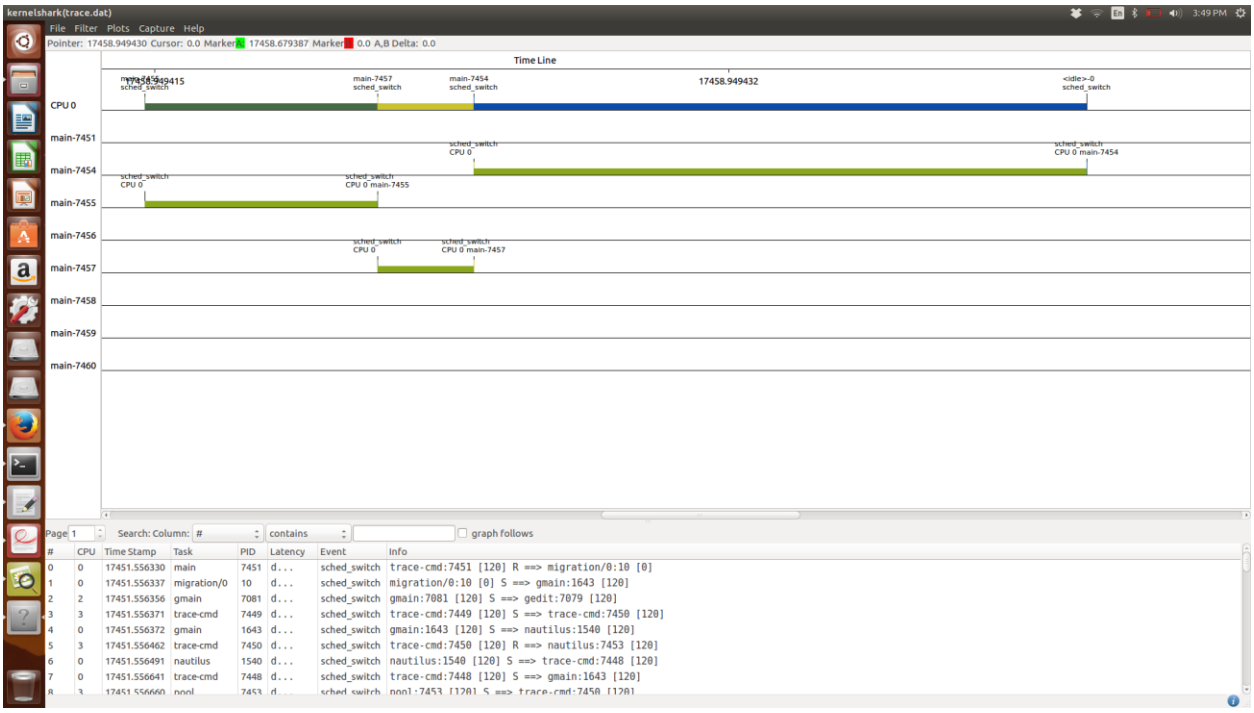
To make the running project we followed the following steps sequentially.

- Made the share queue library.
- Next thing was to create periodic threads both sender thread and receiver by passing the priority and scheduling policy (FIFO for our case) attributes and using clock_monotonic function. Priority of all sender thread is the same while that of receiver thread it is lower than the sender thread. In the periodic function of sender msgID, scrID, and value of PI was passed to the data queue. While for the receiver thread msgID, scrID, and value of PI was read from the data queue.
- Time stamp counter was implemented. For sender thread enqueuing thread was noted and for receiver time dequeue time was noted. Queuing thread was calculated by subtracting the two and dividing with the CPU frequency.
- To create aperiodic thread first thing, we were required to do was to get the permission for the dev directory and to trace down the event number. This initially caused trouble but then we could solve the errors. After detecting the mouse events we created aperiodic threads for both left and right click. Termination was done by using time stamp counter which read the counter values for consecutive left clicks and by subtracting we got the time between two consecutive left clicks.
- PI value was calculated by calling the pi function when every sender thread was created. It generated random numbers between 10 to 50 and returned the PI value which was stored in the data queue.
- By following the steps, we successfully implemented the task.

After the completion of code, we traced the events in the kernel shark by setting the affinity of all threads to CPU 0. Following figures are screenshot of kernel shark running trace.dat file generated.
In the process we can see that always only one thread is active. Moreover, the periodic thread runs periodically while aperiodic thread runs when mouse events are triggered. When a mouse event occurs, aperiodic thread preempts any thread running be it periodic thread or receiver thread. The reason being the priority of aperiodic thread is the highest. Moreover, when a periodic thread tries to call the function which is already being used by some other thread it will wait till the running threads unlocks the mutex and then will start executing,
For all the threads, aperiodic thread is having the highest priority and so it will preempt the thread already running. Also for periodic thread having lower priority it will preempt only the receiver thread if any. Receiver thread is given the lowest priority and it will not preempt any thread.



The graph above shows the FIFO policy of the periodic threads.
We were not able to find an example of preemption are many trials. The following is a general kernelshark output for our program.