

REAL-TIME TRAFFIC MANAGEMENT SYSTEM USING RFID AND RASPBERRY PI

Sai Krishna (824633128) Email – saikrishna96singapati@gmail.com

Kavan Mehta (823999846) Email – mehtakan117@gmail.com

Preetam Bamanalli (824684803) Email- pbamanalli4154@sdsu.edu

Proposed Idea:

In this project we proposed that we will implement real-time traffic management system with the help of RFID readers and tags where the controller can take decisions storing the vehicle details when it's over speeding based on the data collected from the readers in real-time and calculate tolls and will showcase OS concepts of real time scheduling.

Accordingly, we have implemented the real time traffic management system, and created various tasks such as speed calculation, Toll amount calculation and speed Defaulter's calculation by collecting timestamp and the unique id of the RFID tags by the RFID reader.

Accomplishments & Issues faced in this:

Task Developed:

In the task creation the major issue we faced was that we had to interface two raspberry pis as we needed to use 4 RFID (2 RFID on one raspberry pi & 2 RFID on the other) as only 2 RFID can be interfaced to one raspberry pi.

So, when one raspberry pi reads tags (Here cars are referred as tags) from 2nd raspberry pi then it should send details to 1st raspberry pi so that its id and timestamp details should be stored there. All this was achieved using UART interfacing done in python.

Also, the interfacing of two RFID readers on one raspberry pi was difficult as simultaneously reading the two reading was creating a problem as there is no enough online help.

Later on, we realised that SPI can be achieved by toggling the two readers by using pin 24 & 26 So according implemented the code such that we can take data from the two readers using this pin one after the other.

OS Concept developed:

In this project we proposed that we will use OS concept of Scheduling of the above-mentioned task as per the tags received by the readers and then would calculate the waiting time, execution time to find which scheduling policy is more efficient in terms of average waiting and turn around.

Accordingly, we have implemented the OS concept of scheduling the tasks, by implementing two major scheduling policies FIFO and Round Robin. Also, we have used fork for creating process creation.

We have also used multithreading for storing the values from the readers simultaneously.

List of Hardware Components and their significance:

Raspberry Pi 3: Raspberry pi is selected in this as it had SPI which is necessary to interface the RFID readers MFRC522. The os of the raspberry pi is one of the distributions of Linux which has OS libraries to implement scheduling and creation of process.

RFID Readers & Tags: MFRC522 readers were used.

Software Platform/Tools: putty (remote login for Raspberry Pi), Raspbian OS (NOOBS), VNC desktop viewer.

Software implementation:

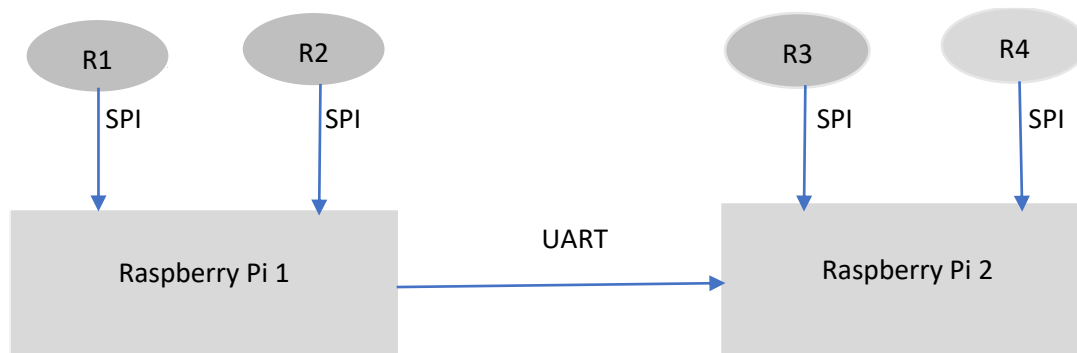
We have used the following libraries in python:

1. MFRC522-python and SimpleMFRC522: These libraries are used to read the ID to the RFID tags using the SPI protocol.
2. OS: This library is used to implement Scheduling and to create process using fork.
3. Time: This is used to get the time when the RFID tag comes near the reader and to get the execution time of the process.
4. Threading: This library was used to create and run two threads.
5. Serial: To send the data from 2nd raspberry pi to the 1st raspberry pi.

Methods used:

1. read: To get the id of the RFID tag.
2. time: To get the time values.
3. sched_setpriority: to set the priority for the scheduling policy.
4. set_scheduler: To set the scheduling policy to either FIFO or RR.
5. get_pid: To get the process ID.
6. fork: To create.
7. Wait: To implement a process which is dependent on the output of child
8. Thread: To create thread
9. Start: To start a thread
10. Join: For the processes main process to wait until the execution of threads are complete.

Approach to the solution:



Firstly, we needed to interface two RFID's with each raspberry pi's. The only libraries that we found useful were MFRC522, which uses the SPI protocol to get and send data to the RFID readers. The library used only one slave pin of the raspberry pi to communicate with RFID reader which was hard coded in the library function. So, we had to make few modifications to the library in order to interface two RFID readers, where in the main function we pass the values for the slave select pin through the class while creating an object of the MFRC522.SimpleMFRC522(0, 0) – This is used to select slave pin- 24 and SimpleMFRC522(0, 1) for selecting slave pin-26. Once the RFID were interfaced with the raspberry pi, we wanted the ID's to be stored in an array so that way we can store the corresponding time values at which the tags arrive at the reader. We created the 2D array to store the time values from four readers.

Secondly, we wanted to read the time values from the 2nd raspberry pi and send it to the 1st one. To implement this, we used the UART communication. We could only send the string from the 2nd raspberry pi. So, we sent the reader from which the time was read, the ID of the tag, and the time in that order. At the 1st raspberry pi we separate the digits using *is digit* function and store in a single dimension array in the same array. If the first digit in

the array was 3, we stored it in the 3rd column. If it was 4, we stored it in the 4th column. And then the time corresponding to the ID was stored.

We wanted to use the threads so that we could store the data from the readers and the 2nd raspberry pi to be stored simultaneously. Thread1 was used to store the data from the readers R1 and R2 and Thread2 was used to store the data from readers R3 and R4.

We wanted three processes to be executed namely speed calculation, speed defaulters & toll calculations in order to create this process we needed to use fork statement.

Creating odd number of processors using fork statement was a challenge to us. We achieved this by the following method. Firstly, we created parent and child using fork statement and then differentiated them using the pids. Then we made the child process to fork once again and these two tasks were further differentiated using pids. And these tasks were killed once job was done.

For scheduling we first set the priority for the scheduling policy. We set the parameters for the scheduler by using sched_param, we passed the priority as parameter. We set the scheduling policy of the tasks by using sched_setscheduler. SCHED_FIFO is used for FIFO and SCHED_RR for Round Robin.

The tasks are executed as and when at least two adjacent time values are available from the readers. The method speed() is called when the data is available and in that method the fork is called to create the processes.

Experimental Design:

1. Success metrics:

Total turnaround time and average turnaround time of FIFO Scheduling policy

| Execution | Task1- Speed | Task2- Toll | Task3- Defaulters |
|---------------|--------------|-------------|-------------------|
| 1st Instance | 1.68359 | 1.17041 | 0.20996 |
| 2nd Instance | 1.5913 | 1.531 | 0.18188 |
| 3rd Instance | 1.73754 | 1.60424 | 0.093675 |
| 4th Instance | 1.55175 | 1.62182 | 0.08129 |
| 5th Instance | 1.58862 | 1.58666 | 0.0979 |
| 6th Instance | 1.59497 | 1.61352 | 0.08471 |
| 7th Instance | 1.72216 | 1.66748 | 0.07983 |
| 8th Instance | 1.6123 | 1.58764 | 0.08715 |
| 9th Instance | 1.72631 | 1.86987 | 0.08837 |
| 10th Instance | 1.90332 | 1.65917 | 0.08569 |
| 11th Instance | 1.5957 | 1.68261 | 0.08642 |
| 12th Instance | 1.77807 | 1.66772 | 0.09399 |
| 13th Instance | 1.76196 | 1.40795 | 0.09838 |
| 14th Instance | 1.5603 | 1.64892 | 0.102 |
| 15th Instance | 1.74145 | 2.39794 | 0.09512 |
| Total | 25.14934 | 24.71695 | 1.5663 |
| Average | 1.67662 | 1.64779 | 0.10442 |

Total turnaround time and average turnaround time of Round Robin Scheduling policy

| Execution | Task1- Speed | Task2- Toll | Task3- Defaulters |
|---------------|--------------|-------------|-------------------|
| 1st Instance | 1.53417 | 1.63159 | 0.15966 |
| 2nd Instance | 1.48291 | 1.64526 | 0.0957 |
| 3rd Instance | 1.44409 | 1.36303 | 0.13452 |
| 4th Instance | 1.45898 | 1.35815 | 0.08935 |
| 5th Instance | 1.48266 | 1.35864 | 0.08935 |
| 6th Instance | 1.72778 | 1.51684 | 0.08593 |
| 7th Instance | 1.67456 | 1.57324 | 0.10278 |
| 8th Instance | 1.67749 | 1.44482 | 0.08642 |
| 9th Instance | 1.63793 | 1.50463 | 0.08325 |
| 10th Instance | 1.51196 | 1.52148 | 0.08081 |
| 11th Instance | 1.72827 | 1.69458 | 0.08764 |
| 12th Instance | 1.82104 | 1.53613 | 0.08789 |
| 13th Instance | 1.68188 | 1.40332 | 0.08377 |
| 14th Instance | 1.53076 | 1.74853 | 0.10131 |
| 15th Instance | 1.72735 | 1.44532 | 0.08154 |
| Total | 24.12183 | 22.74566 | 1.44992 |
| Average | 1.608122 | 1.5163 | 0.09661 |

From the above tables we can see that the average turnaround time is less in the RR scheduling policy is less than the FIFO as expected, Since the RR reduces the average waiting time.

2. Experiments and Tests performed:

We initially interfaced single RFID reader to one raspberry pi in order to check how the RFID readers read the tag values so that we can know the length of the id of the tag.

After that we interfaced 2nd RFID reader on the same raspberry pi by using pin 24 (GPIO 8 SPI_CEO_N) for the 1st RFID reader and pin 26 (GPIO 7 SPI_CE1_N) for 2nd RFID reader and checked if the id values are read by the raspberry pi. Same was applied to the 2nd raspberry pi.

We checked if the time was recorded when the tags were read by the RFID readers.

We checked if the time was stored into the array from the readers R1 and R2.

We checked if the string of the reader, tag id and time from the second raspberry pi was send to the 1st raspberry pi using the UART communication.

We checked if the string from the second raspberry pi the values of the string from the second raspberry pi were separated based on the values and stored in the single dimensional array.

We checked if the time values from the separated string were stored in a 2D array based on the comparison of id and the reader value.

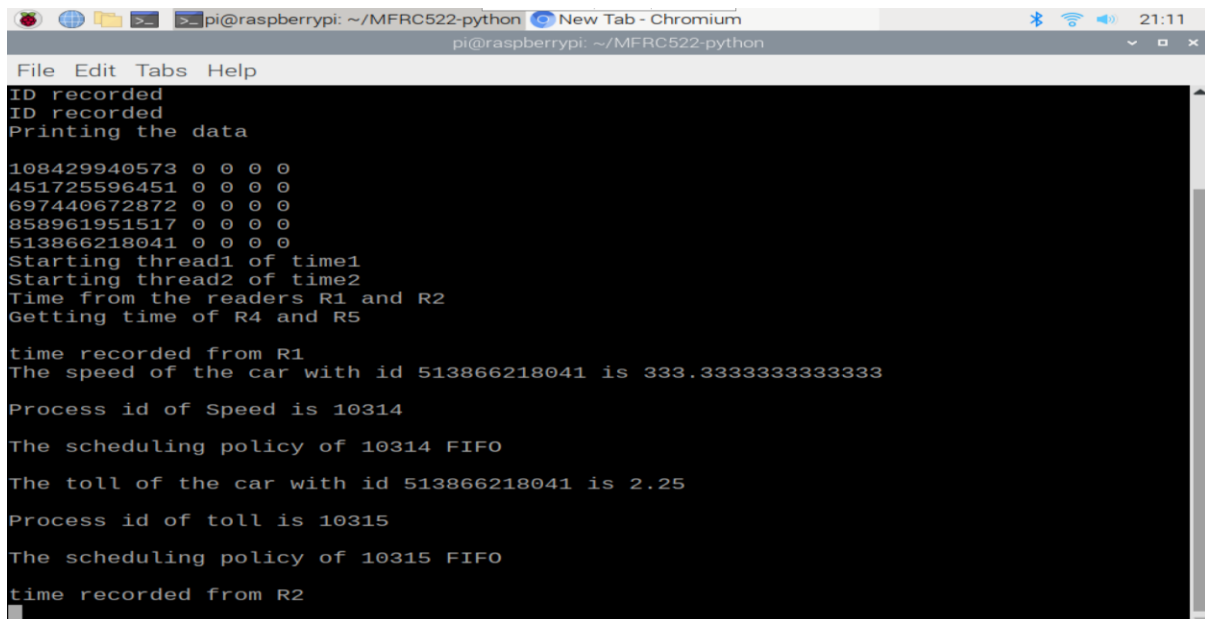
```
513866218041 72565 72566 72608 72610
451725596451 72576 72577 72597 72598
108429940573 72573 72574 72601 72602
858961951517 72569 72570 72605 72606
697440672872 72562 72563 72593 72595
pi@raspberrypi: ~/MFRC522-python $
```

We checked if the tasks speed toll and speed defaulters were executed when the time values are available.

For scheduling of task, we wanted to perform FIFO and Round robin so we first created dummy task and then used that task to perform scheduling as we have had no idea on how real time os scheduling will work on tasks. This approach acquainted us with the scheduling policy working on dummy task and made it easier to debug our mistakes. It made us took more time as we were brain storming it and trying different method implementing scheduling policy.

We checked the scheduling policy of our task by using the function get scheduler, this function returns 1 if the scheduling policy were FIFO and 2 when it was RR so based on these values, we confirmed whether the scheduling policy were RR or FIFO.

FIFO:



```
File Edit Tabs Help
ID recorded
ID recorded
Printing the data
108429940573 0 0 0 0
451725596451 0 0 0 0
697440672872 0 0 0 0
858961951517 0 0 0 0
513866218041 0 0 0 0
Starting thread1 of time1
Starting thread2 of time2
Time from the readers R1 and R2
Getting time of R4 and R5

time recorded from R1
The speed of the car with id 513866218041 is 333.3333333333333
Process id of Speed is 10314
The scheduling policy of 10314 FIFO
The toll of the car with id 513866218041 is 2.25
Process id of toll is 10315
The scheduling policy of 10315 FIFO
time recorded from R2
```

Round Robin:

```
192.168.0.15:1 (pi's X desktop (raspberrypi:1)) - VNC Viewer
pi@raspberrypi: ~/MFRC522-python
File Edit Tabs Help
The total execution time of tasks is 0.093994140625
Time from R4 recorded
The speed of the car with id 513866218041 is 500.0
Process id of Speed is 8197
The scheduling policy of 8197 Round Robin
The total execution time of speed task is 1.44921875
The toll of the car with id 513866218041 is 2.25
Process id of toll is 8198
The scheduling policy of 8198 Round Robin
The execution time of Toll task is 1.50732421875
The total execution time of speed default task is
0.098388671875
The total execution time of the tasks is 0.098388671875
Printing the data
513866218041 72565 72566 72608 72610
451725596451 72576 72577 72597 72598
108429940573 72573 72574 72601 72602
858961951517 72569 72570 72605 72606
697440672872 72562 72563 72593 72595
pi@raspberrypi:~/MFRC522-python $
```

3. Results and their significance:

In this project we were able to record the time at which the car arrives near the RFID reader and take the time difference between successive readers and calculate the speed. We were able to calculate the toll based on event when a car arrives near the RFID reader. From the success metrics, experiments and tests performed we have verified each and every step in the process of the solution and also, we have tested it altogether. We were able to achieve the desired scheduling policy for the tasks.

4. Missing Milestones:

1) SJF scheduling: We weren't able to complete this milestone as there were no predefined functions in the os libraries. Since SJF is non-pre-emptive we thought we will implement it using sched_FIFO by the changing the priority 0 when we did that the order of execution was not in sequence. When process 2 had higher priority than process 1, the process 1 codes were executing before process 2.

2) Maximum Lateness: Since there is no specific deadline for the task and also the tasks were small and the execution of each task were very small.

5. Challenges faced:

- 1) Connecting two RFIDs to the raspberry pi using the two SPI interfaces.
- 2) Connecting the 2 raspberry pis.
- 3) Changing the time stamp format given by the RFID reader to make it useful for our calculations.
- 4) Creating odd no. of process only using fork statement
- 5) Identifying that print () was buffered

- 6) Accessing raspberry pi on laptop
- 7) Changing the library of the RFID reader to interface 2 RFID's.

6. Possible extensions to the project:

We can extend this project by adding more functionality with the help of this implementation:

Some of the extensions we thought of are:

- 1) Accident detection
- 2) Congestion control
- 3) Creating a database which can store all this data for security purpose.
- 4) Sending an online speed ticket and toll levied through email.
- 5) Analyzing traffic patterns with the data stored.

References:

- 1) <https://www.raspberrypi.org/forums/viewtopic.php?p=545280>
- 2) <https://www.deviceplus.com/connect/integrate-RFID-module-raspberry-pi/>
- 3) <https://pimylifeup.com/raspberry-pi-RFID-rc522/>
- 4) <https://www.instructables.com/id/Read-and-write-from-serial-port-with-Raspberry-Pi/>
- 5) https://python.hotexamples.com/examples/os/-/sched_setscheduler/python-sched_setscheduler-function-examples.html
- 6) <https://www.geeksforgeeks.org/creating-child-process-using-fork-python/>
- 7) <https://www.geeksforgeeks.org/multithreading-python-set-1/>
- 8) <https://github.com/mxgxw/MFRC522-python>
- 9) <http://man7.org/linux/man-pages/man7/sched.7.html>