1. Problem Statement

The task was to develop a software tool that takes the program as input and outputs all valid traces

2. The algorithm used and approach

We have converted the input program into the graph, then we have used the Topological sorting algorithm to generate all possible traces. Then we removed all the traces having the same rf and ws relations to generate all the valid traces, for this we used the set to find the unique traces.

1. We have taken the input as the command line argument.
2. A graph of adjacency list is created as follows:
    - 2.1. Each vertex corresponds to the instructions of the input program
    - 2.2. Each edge between sequential instructions corresponds to the po relations.
    - 2.3. We added global variables initialization as the vertices of the graph and these vertices have an edge to the first instruction of each process to ensure that initialization happens before any process execution.
    - 2.4. Encoding and Decoding of vertices
        - 2.4.1. **Encoding**: According to the input constraint that there are only 10 processes possible, so we have assigned a unique number for all the instructions and to differentiate between the instructions we have added the process number and line number in front of each instruction
        - 2.4.2. **Decoding**: We have created the inverted_index mapping for each unique number which is mapped to the instructions.
3. We have used a topological sorting algorithm and using it we have generated all possible traces
4. rf and ws relations for each trace are generated as follow:
    - 4.1. We created a function get_relation which takes the trace as input parameter and returns the rf and ws relations corresponding to each trace
    - 4.2. Basically, we have traversed each trace, created some buffers to store the previous writes.
    - 4.3. Then when we get any “read” instruction then we simply add this instruction and buffered instruction as an rf relation
    - 4.4. If we get “write” instruction then we see whether this write is from another process or not, if yes then we created the ws relation between these two.
5. Redundant traces (which have same rf, ws) are removed as follows:
    - 5.1. We have created a set
    - 5.2. First, we sorted rf and ws relations

5.3. Then we converted the list of relations into a string form
5.4. Then we see whether this string is already present in the set or not, if yes then this relation is redundant and do not include this trace and relations into the answer

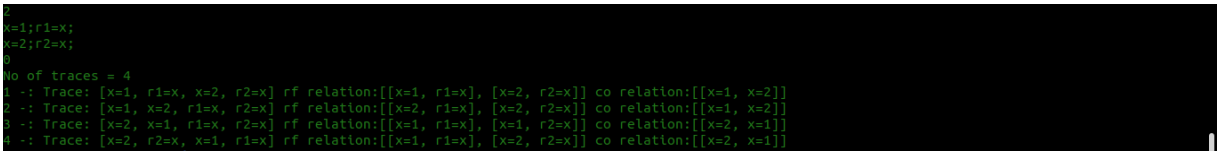
6. Assertion check is carried out in the following manner:
6.1. Store the values of local registers into an array
6.2. Create a temporary python file
6.3. Now, write those values to a temporary python file as python variables initialization
6.4. Write the assertion statement as python's built-in assert method
6.5. Now, execute this python file and read the output
6.6. If it generates python's Assertion Error, we will consider the corresponding trace as the trace violating the same.

3. How to run the file

open the terminal
go to the directory where this file is present
Type: python3 trace.py
Then input will be asked, give input on the command line as shown below:
2
x=1;r1=x;
x=2;r2=x;
0

4. Result/Output

Output1:



Output2: