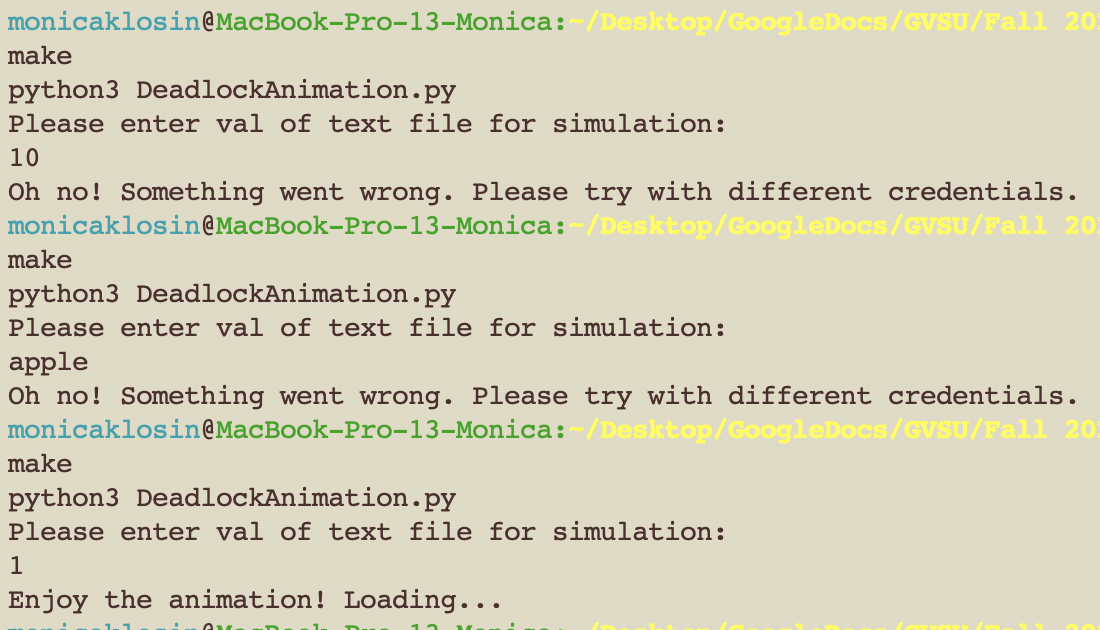
**CS 452 Project -- Resource Manager**

Monica Klosin

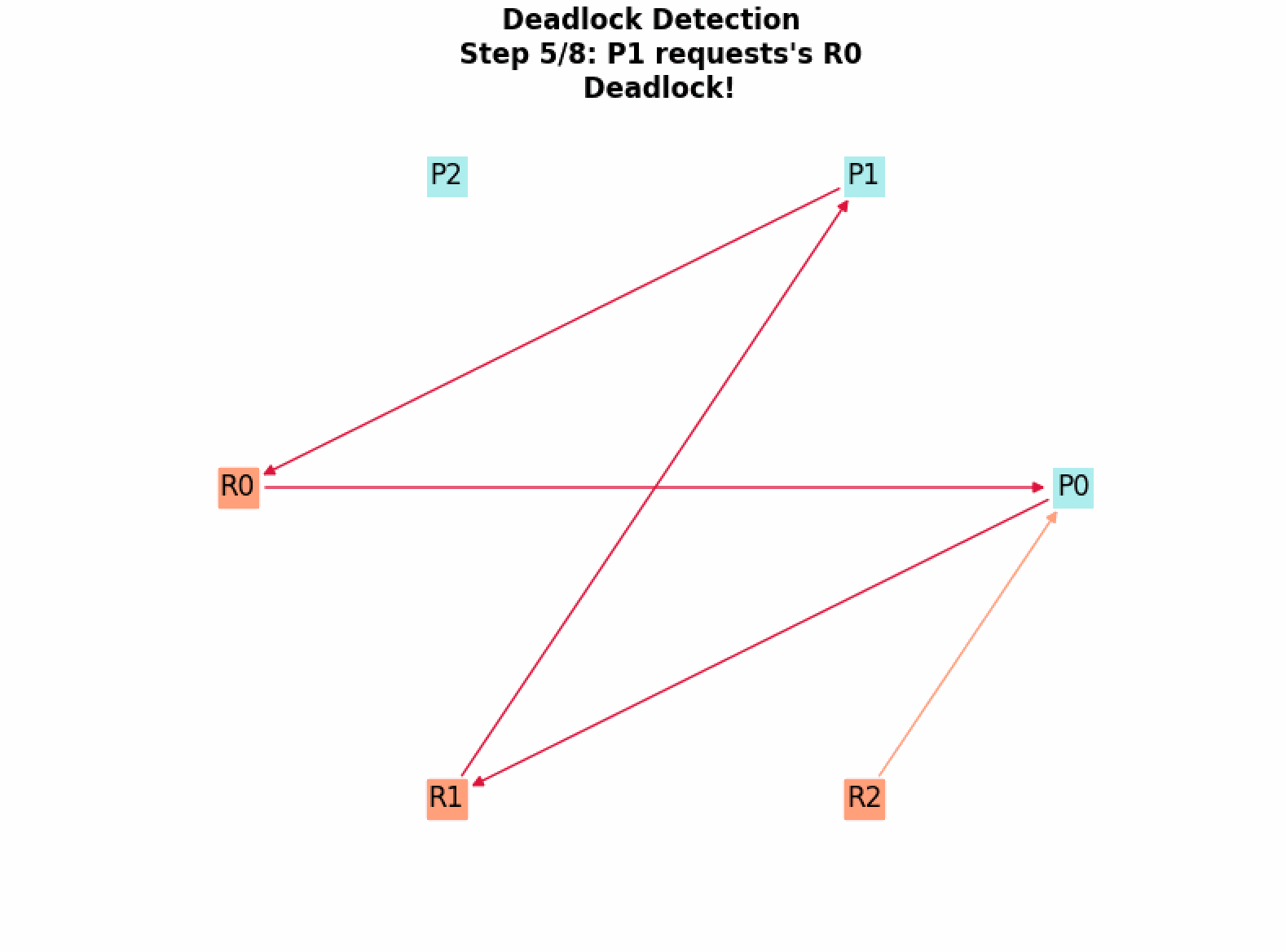
**Repo to see animated simulation (in README):** <https://github.com/klosinm/Operating-Systems>

**To Run:**

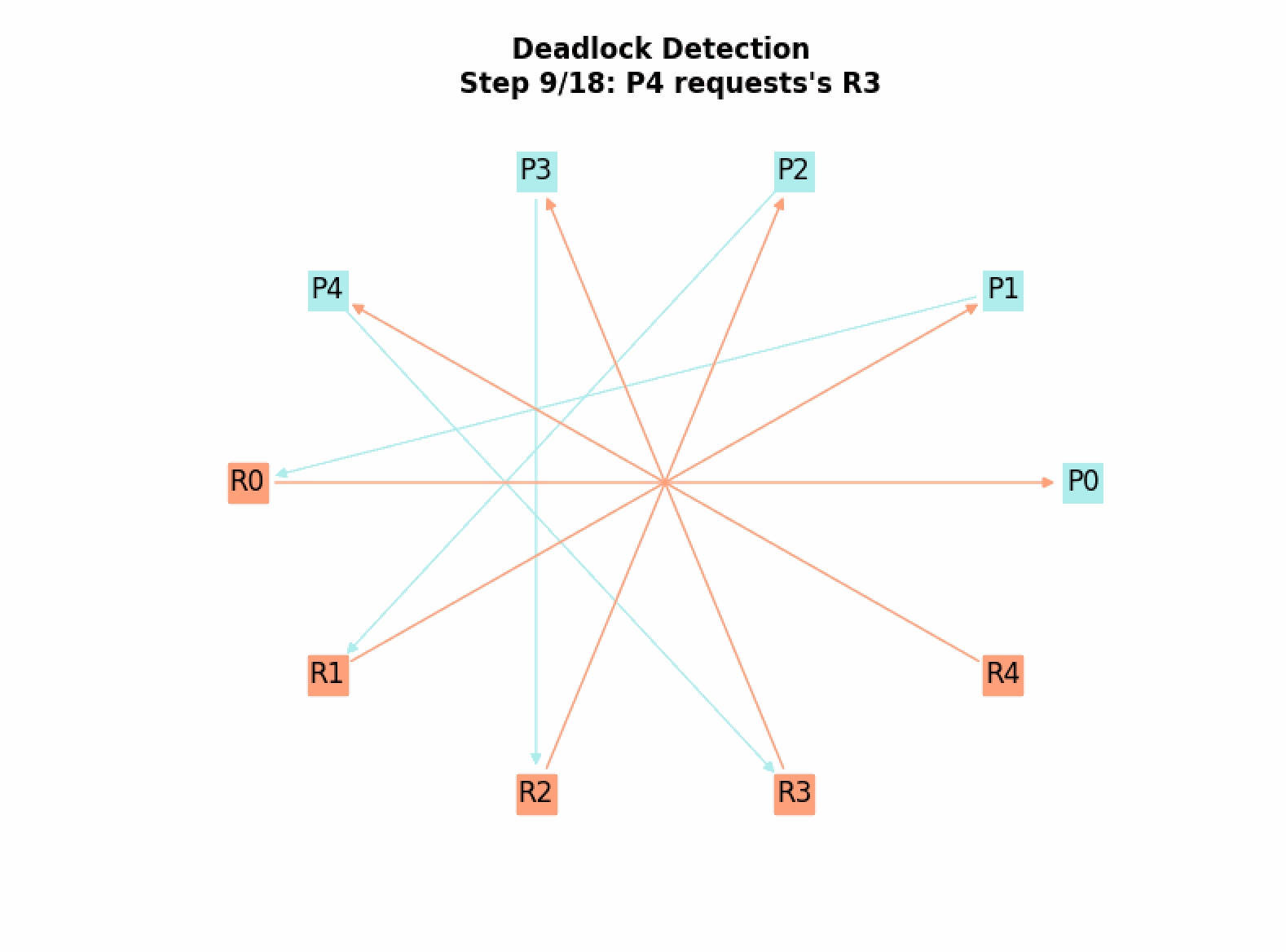
1. in directory, type *make*
2. Since all file names for this program are written as “scenario-{*number*}.txt” , user then provides an integer to fill in for {*number*} to choose which file they want the simulation to run with
   1. screenshot below shows test cases for incorrect and correct user input



**Screenshots of Program:**



ran with edited scenario-2.txt



ran with scenario-3.txt

**Extra Credit:**

I decided to try to implement the detect-and-recover policy functionality.

*If there is deadlock detected, you kill the processes involved in the deadlock one by one. After each deletion of one process, you check for deadlock again until there is no more deadlock and let the program continue as normal.*

**Strategy:**

To implement (plan):

* Take the cycle created via the list(nx.simple\_cycles(G))
  + set this command to variable T (T = list(nx.simple\_cycles(G)))
* grab the first created edge part of the cycle and remove it via remove\_edge
* notify the simulation that we are removing this edge, and update the array that indicates which processes hold what resources (processHolder), the array that indicates if a resource is held (resourceHeld) and the array that indicates which PID are waiting for that resource (resourceWanted)
* append this killed process to the end of the request line to run it last
  + try to allow this process to only run if the resource it wants is free

**Outcome:**

I did not visualize the detect and recover simulation via matplotlib, I instead just displayed in terminal.

I also was unable to do the very last sub-bullet of my plan, which would be to allow the program to only run if the appended step if the recourses the process wanted were free.

With scenario-2.txt:

Based on what I coded, this is the outcome in a drawing:



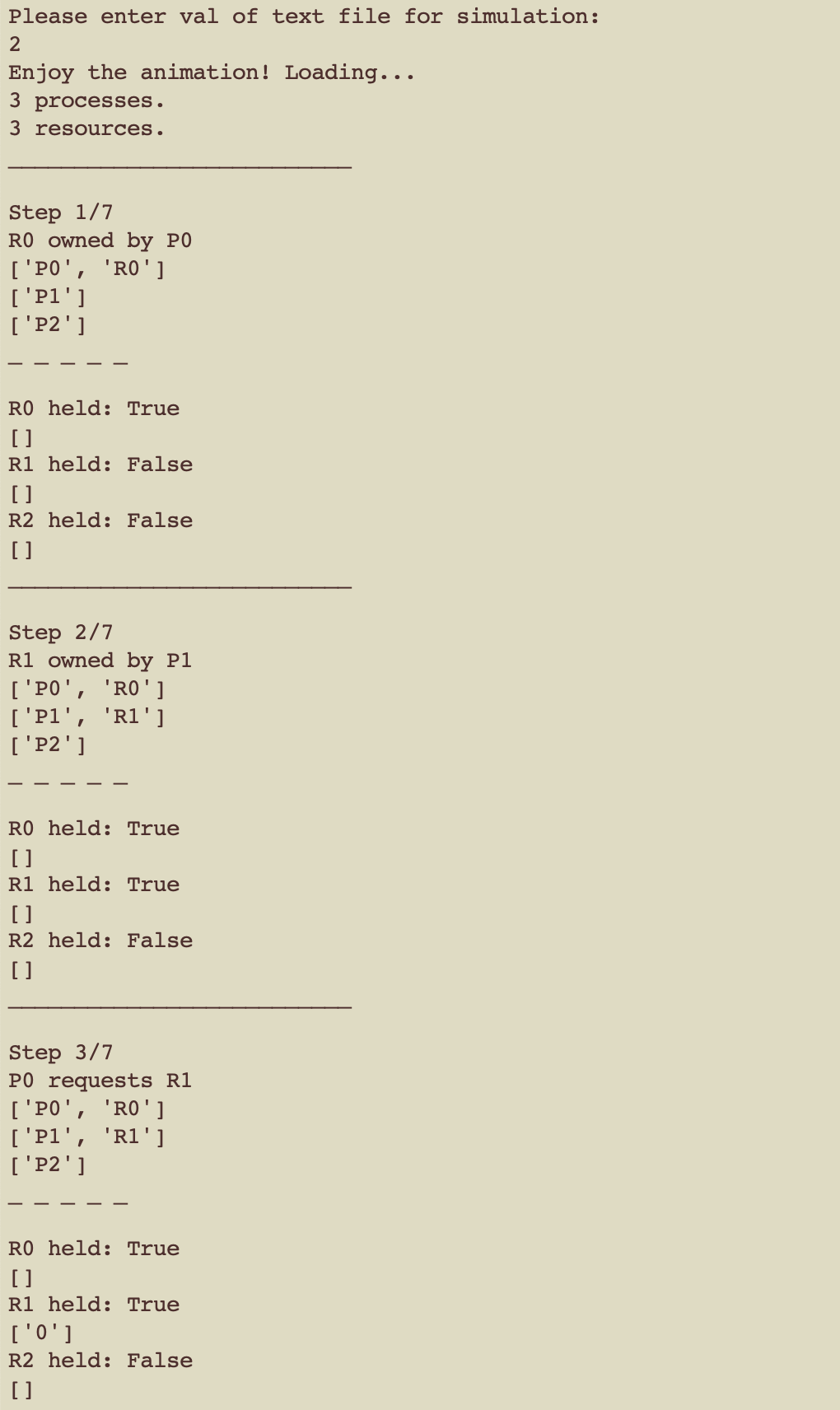
But it should be this,



because the appended step should only be “reset” after the resources (R0) it (P0) needs are available.

Since P1 still owns R0, P0 should only request R0 after R0 is freed from P1.

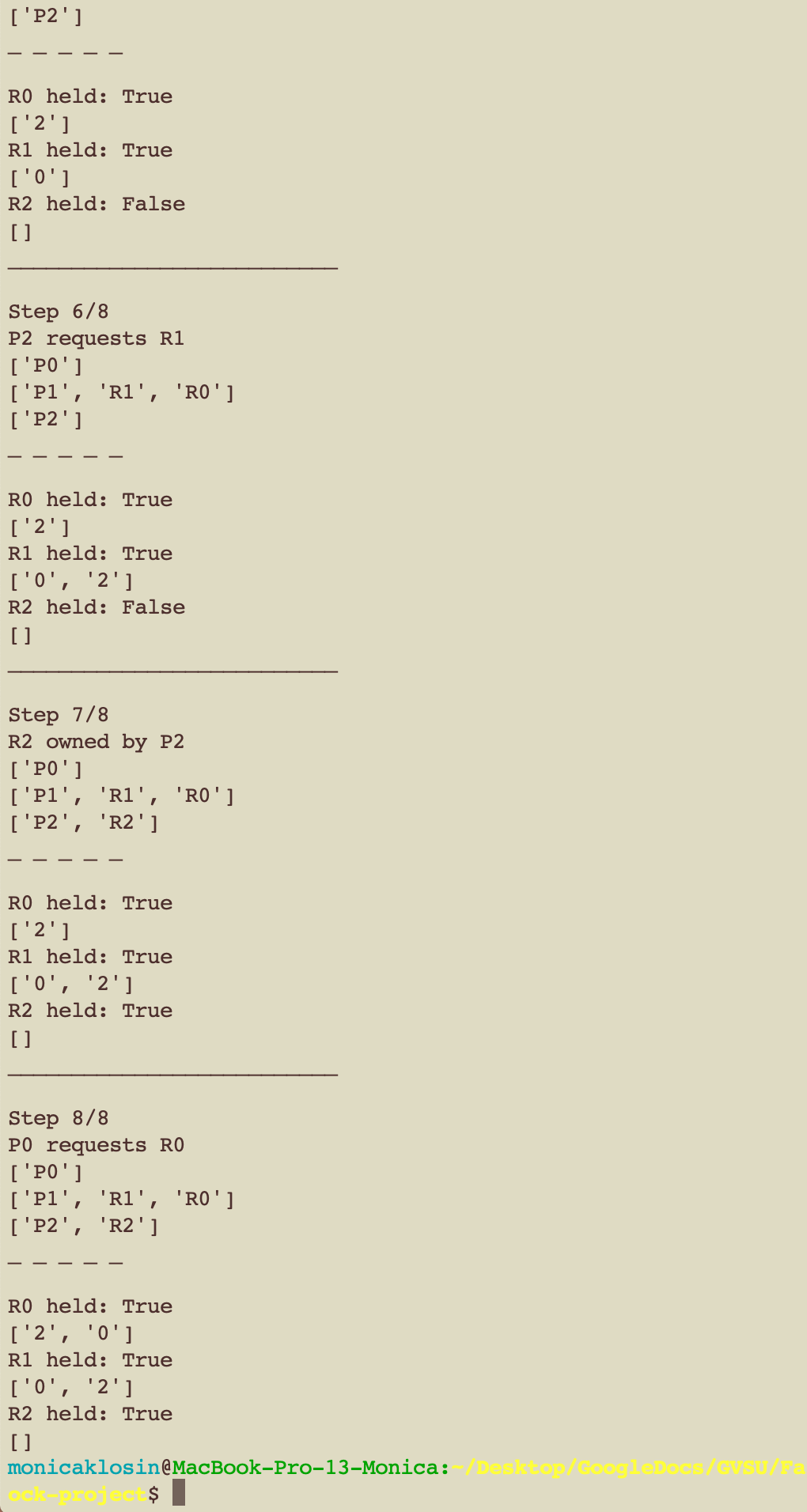
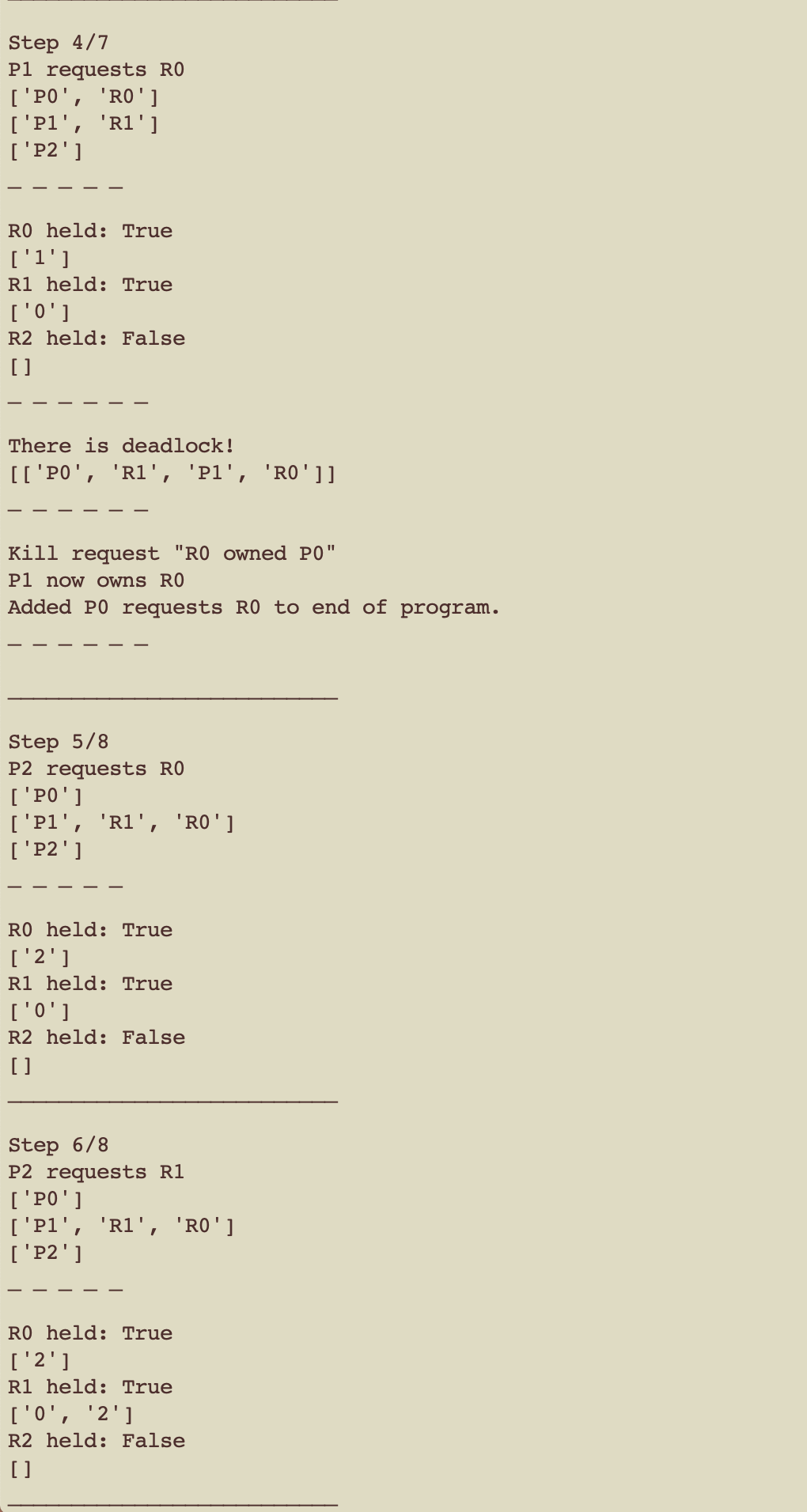
Terminal output of Deadlock Detection and Recovery:



List the Resources, T/F if they are held by a Process, and an array below listing the PID(s) that are requesting it (P -> R)

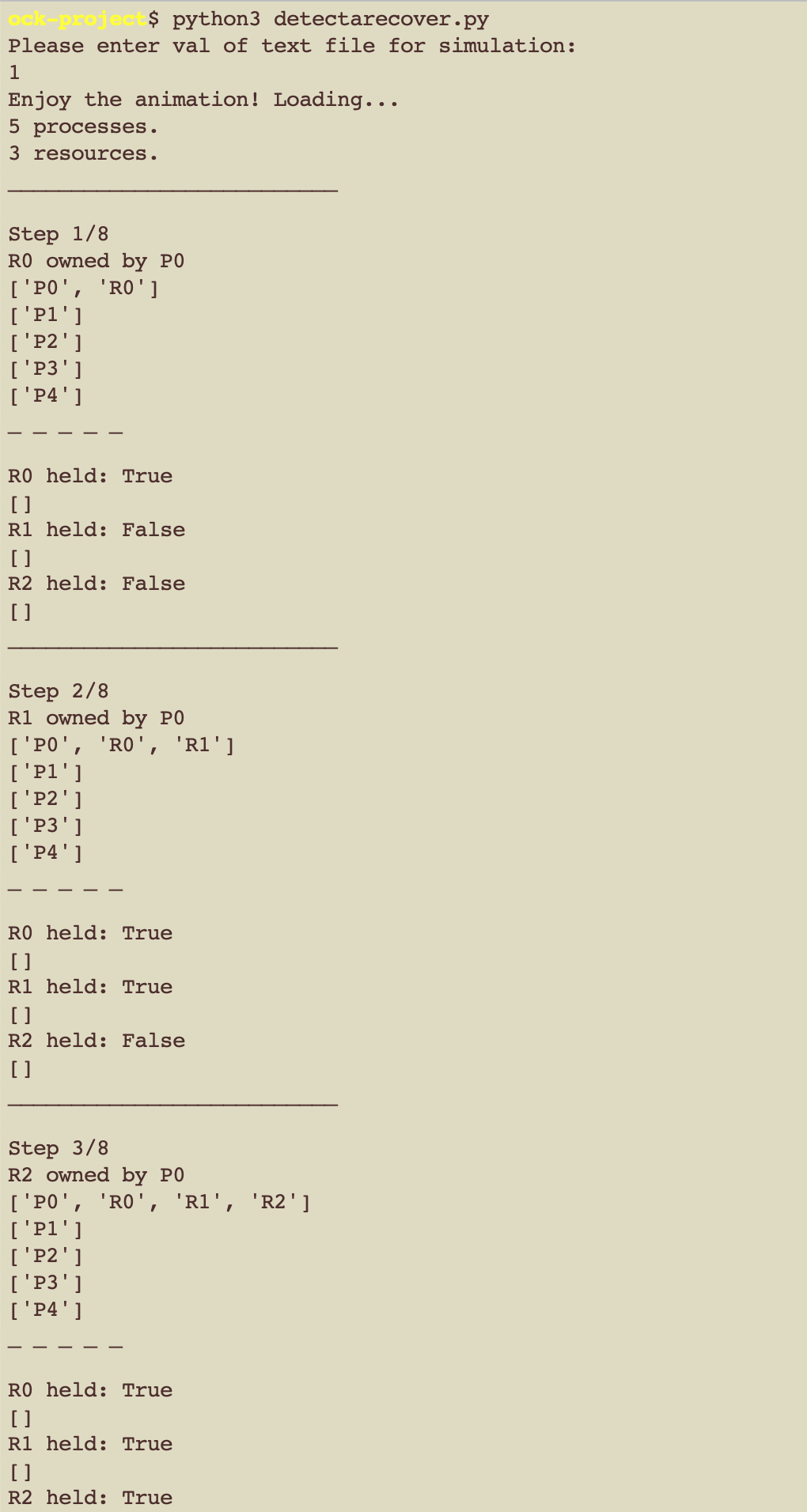
List the Processes in the program, followed by the Resources they own

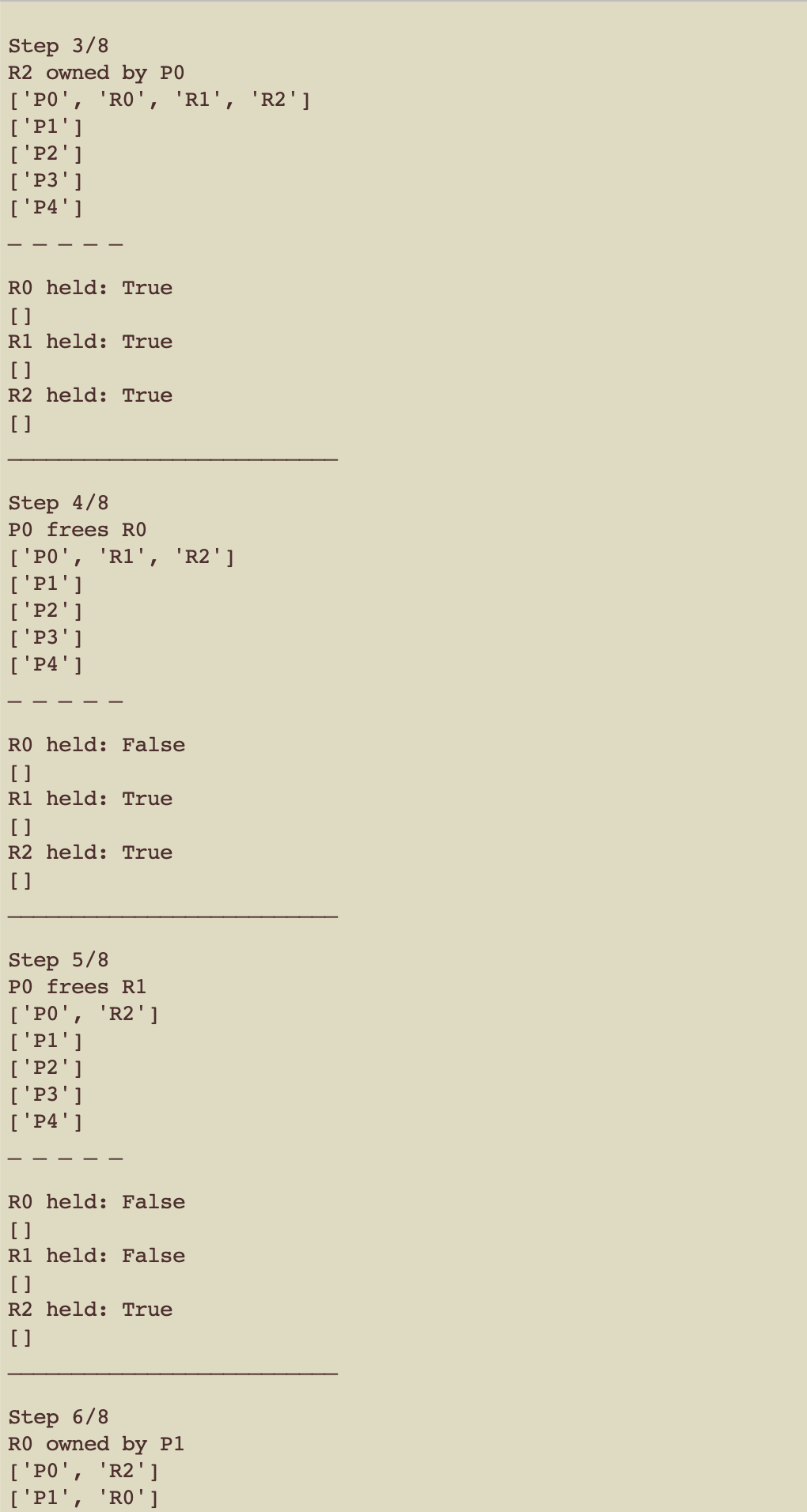
(R -> P)

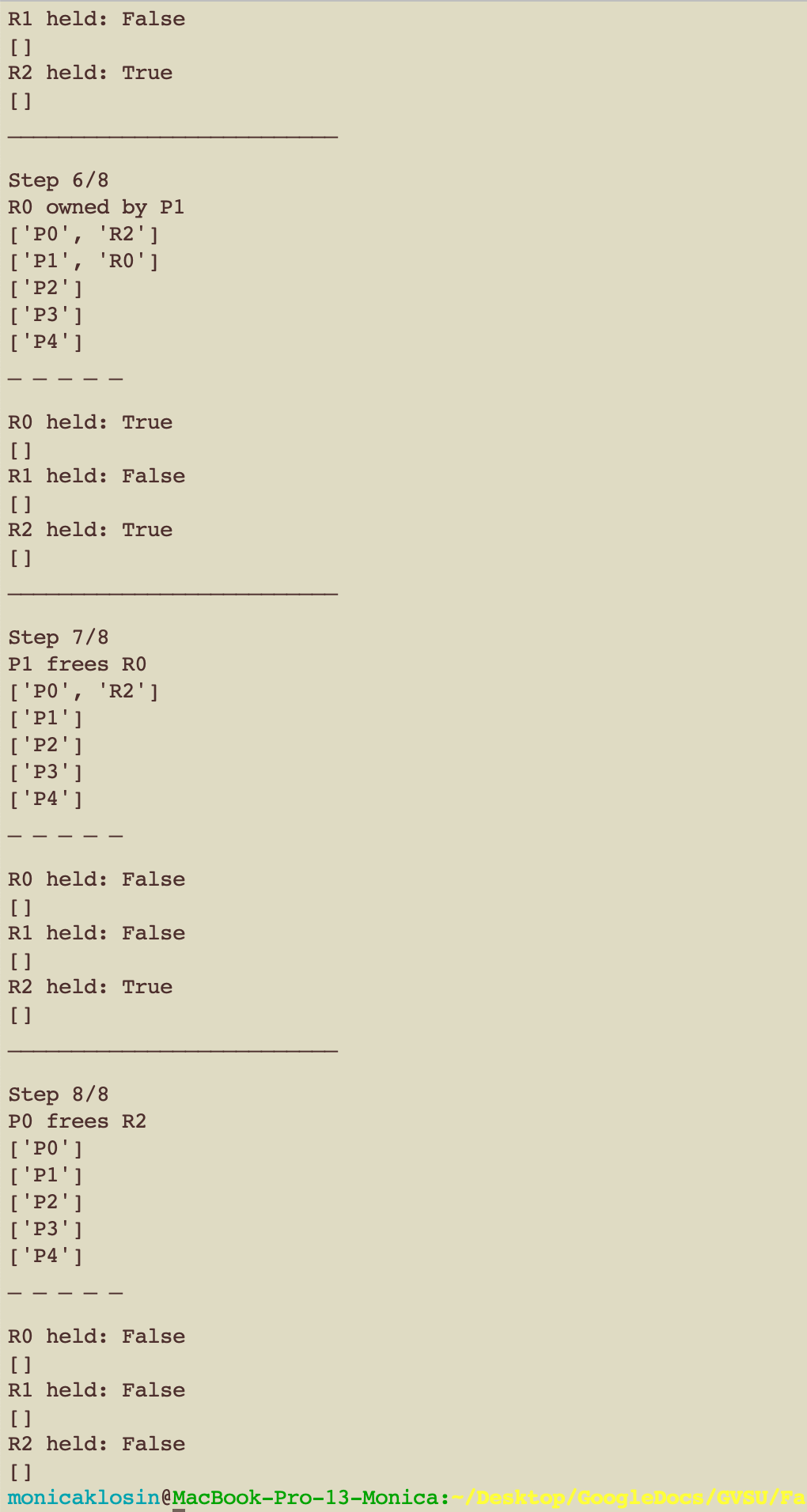


(The simulation is a lot prettier)

Same program, but running with program that doesn’t have deadlock (scenario-1.txt):







Wahoo! Done ☺