

1 Financial crisis (4 pts.)

Using the tabular trace format and the “e-commerce” sample code found in the lecture slides, show how the presence of a privileged citizen “interrupt” can cause deflation in the general economy.

Execution Trace

time	Regular customer	Interrupt customer	store->cash
0	cash = store->cash		1500
1	cash += 50		1500
2	wallet -= 50		1500
3		cash = store->cash	1500
4		cash += 100	1500
5		wallet -= 100	1500
6		store->cash = 1600	1600
7	store->cash = 1550		1550

The problem is that the two wallets have spent a total of \$150 but the store is only \$100 richer. Thus the total supply of money in the economy has lost \$50 (and perhaps more to the point, the store in particular has lost that money).

Note that, in order to receive full credit, your solution must make it clear which “instructions” are executed when with respect to each other, and must also show enough data values to justify control-flow choices you make. For problems involving looping, you must make it clear when a loop is run multiple times, and why.

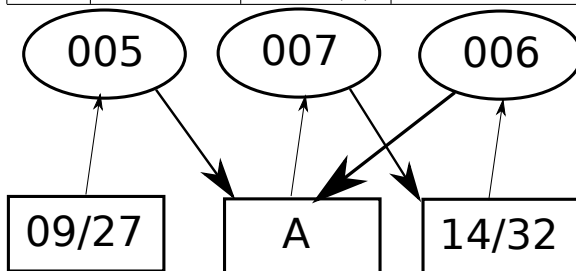
2 Runways (10 pts.)

2.1 5 pts

Explain a genuine deadlock situation which can arise.

Execution Trace

time	Flight 005	Flight 006	Flight 007	Comment
0	land(27)			taxi end of 09/27 contains a plane
1		land(32)		taxi end of 14/32 contains a plane
2			taxi_out(A,14)	stalls because of 005, 006
3	taxi_in(A)			must wait while 007 present
4		taxi_in(A)		must wait while 007 present



Note that other formulations are possible. For example, you could consider Taxiway A to consist of three segments, A1, A2, and A3. Deadlock can be shown (more easily) with respect to Taxiway B.

Also note that you should not have one giant object representing all runways at the airport. While that is a feature of *this* airport, it does not scale. Many airports have more than two runways; while two or three runways commonly cross, it is less common for four or more to all cross.

2.2 5 pts

Can you state a concise anti-deadlock rule which should be added to the training handbook for controllers working at this airport? Or can you otherwise characterize which patterns of orders the controller must not issue?

The controller is well-advised to do all allocations for a flight “at once”: for a plane leaving via “take-off(32),” the controller should allocate Taxiway B and Runway 14/32 as a unit. Then the flight can “proceed to completion” and release both resources. This will result in some resources being idle (if, for example, the flight needs to sit at the end of Taxiway B for a while getting de-iced, the runway will be idle), but because the “processes” in this example can’t roll back (turn around or reverse), and because no process needs many resources (one of each kind), this seems like a reasonable approach.

Note that *allocating* all-at-once does not mandate *releasing* resources all-at-once. Releasing resources as soon as you’re (genuinely) done with them is productive.