```
1
2   # This assumes that there is one barber,
3   # multiple waiting seats and multiple
4   # customers. Also assumes that its the
5   # barber that moves around to cut hair
6
7   # stores the number of customers ready and
8   # sitting in a chair waiting for a hair cut
9   counting_semaphore customerReady
10
11  # protects the chairs available from race
12  # conditions.
13  # we cant use a counting semaphore here
14  # because we cant bind how many threads
15  # are waiting. Though I may need more research
16  # to verify this
17  mutex chairs
18  chairs_available = 3
19
20  # stores if the barber is done cutting hair
21  # used to signal to the customer that the
22  # barber has finished cutting hair
23  binary_semaphore cut
24
25  def barber_function():
26      while True:
27          wait(customerReady)  # Wait until a customer is ready
28          cutHair()  # cut their hair
29
30
31  def customer_function():
32      leave = False
33
34      # check if there are available chairs
35      # to sit in
36      lock(chairs)
37      if (chairs_available == 0):
38          leave = True
39      else:
40          chairs_available -= 1
41      unlock(chairs)
42
43      # leave if no seats are available
44      if (leave):
45          return
46
47      # increment semaphore signaling another
48      # customer is ready to get a hair cut
49      signal(customerReady)
50      getHairCut()
51
52      # lock the chairs again and set it
53      # available for other people
54      lock(chairs)
55      chairs_available += 1
56      unlock(chairs)
57
58  # cut hair, which takes 2 secs
59  # then signal to customer that
60  # we have finished cutting hair
61  def cutHair():
62      sleep(2)
63      signal(cut)
64
65  # wait for hair to be cut
66  # called by customer
67  def getHairCut():
68      wait(cut)
```

```
# the signal that this thread is ok
# to board
conditional_var waiting_hackers
conditional_var waiting_employees

# this mutex protects the num_* variables
mutex num

# the current number of hackers and employees
# waiting to board
num_hackers
num_employees

# This variable is needed because
# there exists a execution path where the
# boart driver thread is waiting for the
# num_* counters to decrement and another
# hacker or employee joins the beach. It
# cant differentiate them and it waits
# for the number to decrement, which
# it never does
counting_semaphore num_on_boat

def RowBoat():
    thread.exit()

#
def HackerArrives():
    # This variable stores if this thread is
    # boat driver or not. The boat driver
    # signals the other threads who is able
    # to board and starts the boat motor once
    # everyone is on board
    captain = False

    lock(num)
    # The newcoming thread checks if the boat can take
    # off. If there are 4 of us (num_hackers >= 4) then
    # we can signal 3 of our hacker friends and take off
    # also if there are more than 2 employees and more
    # than 1 other hacker we can also take off. If we
    # find either of these conditions, we have become
    # the boat driver and will signal and wait for the
    # people to board the boat. If we dont then we just
    # increment the waiting counter and wait on the
    # conditional variable
    if (num_hackers >= 3):
        # assume captains duty
        captain = True

        # wait until everyone is on the boat. By setting
        # the num_on_boat variable, we are essentially
        # locking the boat and preventing anyone from
        # assuming captains duties while we are boarding
        num_on_boat = 3

        # signal 3 of our hacker friends to board
        # the boat
        for i in range(3):
            signal(waiting_hackers)

        # wait on the semaphore to reach 0
        wait(num_on_boat)

    elif (num_employees >= 2 and num_hackers >= 1):
        captain = True

        num_on_boat = 3

        for i in range(1):
            signal(waiting_hackers)
        for i in range(2):
            signal(waiting_employees)

        wait(num_on_boat)

    else:
        num_hackers += 1
    unlock(num)

    if (captain):
        RowBoat()
    else:
        wait(waiting_hackers)
        num_on_boat -= 1

# the code is the same as above, except *_employees is
# swapped with *_hackers
def EmployeeArrives():
    captain = False

    lock(num)
    if (num_employees >= 3):
        captain = True
        num_on_boat = 3

        for i in range(3):
            signal(waiting_employees)

        wait(num_on_boat)

    elif (num_hackers >= 2 and num_employees >= 1):
        captain = True

        num_on_boat = 3

        for i in range(1):
            signal(waiting_employees)
        for i in range(2):
            signal(waiting_hackers)

        wait(num_on_boat)

    else:
        num_hackers += 1
    unlock(num)

    if (captain):
        RowBoat()
    else:
        wait(waiting_employees)
        num_on_boat -= 1
```

If the machine never had a deadlock, the machine would complete 5000 jobs and would net the company $2 * $5'000 = $10'000

Because of the deadlock, 20 jobs have to be rerun which is about $60 (20 jobs * 1.5 lost during failure (they are half done) * 2 lost during rerun). This cost is negligible. The real cost is the salary of the employee. Which at minimum wage (7.25) is costing the company $1'160 per month, assuming 40-hour work week and 4 weeks a month.

If the employee is fired and the dead lock is mitigated using a algorithm, All the jobs still get completed, but without the $60 + $1'160 cost of the salary and the job restart cost.

Yes, the company should fire the employee and setup a deadlock algorithm