```
import random
import threading
import time
import os
import sys

def show(mess):
    os.write(1, mess + "\n")

def down(sem):
    sem.acquire()

def up(sem):
    sem.release()

class Inspection():

    # Questa classe, di cui viene creata una sola istanza, assegnata
    # ad una variabile globale INSPECTION, serve a gestire la
    # sincronizzazione tra il manager (unico) e i vari camerieri. La
    # classe contiene sia il dato relativo al risultato dell'ispezione
    # (passed) sia i due semafori per la sincronizzazione (requested,
    # finished) che il semaforo per il controllo di accesso al manager
    # (lock).

    def __init__(self):
        self.passed = False
        self.requested = threading.Semaphore(0)
        self.finished = threading.Semaphore(0)
        self.lock = threading.Semaphore(1)

class Line():

    # Questa classe, di cui viene creata una sola istanza, assegnata
    # ad una variabile globale LINE, serve a gestire non la
    # sincronizzazione tra i vari clienti ed il cassiere ma tra i vari
    # clienti che si vogliono mettere in coda.  Infatti la
    # sincronizzazione con il cassiere Ã¨ implicita nel fatto che c'Ã¨
    # una coda (gestita con il vettore di semafori CUSTOMERS).

    def __init__(self):
        self.number = 0
        self.requested = threading.Semaphore(0)
        self.customers = [threading.Semaphore(0)] * 10
        self.lock = threading.Semaphore(1)

INSPECTION = Inspection()
LINE = Line()

def manager(tot_cones):

    global INSPECTION

    approved_count = 0
    inspected_count = 0
    while approved_count < tot_cones:
```

```
        down(INSPECTION.requested)        # wait for inspection request
        inspected_count += 1
        mess = "manager: inspection %d" % inspected_count
        show(mess)
        INSPECTION.passed = random.choice(range(10)) > 0
        if INSPECTION.passed:
            approved_count += 1
        up(INSPECTION.finished)           # signal clerk

    time.sleep(1)
    mess = "manager leaves: %d approved %d rejected" % (
        approved_count, inspected_count - approved_count)
    show(mess)

def make_cone():
    pass

def clerk(id, customer, cone, done_semaphore):

    global INSPECTION

    passed = False
    while not passed:
        mess = "clerk %2d: making cone %d for customer %d" % (
            id, cone, customer)
        show(mess)
        make_cone()
        down(INSPECTION.lock)             # enter critical region
        up(INSPECTION.requested)          # ask for inspection
        down(INSPECTION.finished)         # leave office
        passed = INSPECTION.passed
        if passed:
            mess = "clerk %2d: cone %d for customer %d passed" % (
                id, cone, customer)
            show(mess)
        else:
            mess = "clerk %2d: cone %d for customer %d REJECTED" % (
                id, cone, customer)
            show(mess)
        up(INSPECTION.lock)               # leave critical region
    up(done_semaphore)                    # signal customer

def browse():
    pass
def walk_to_cachier():
    pass

def customer(id, cones_count):

    global LINE

    clerk_done = threading.Semaphore(0)
    browse()
    mess = "customer %d: asking for %d cones" % (
        id, cones_count)
    show(mess)
    for c in range(cones_count):          # "create" clerk
```

```
        t = threading.Thread(target=clerk,
                             args=(id * 10 + c, id, c, clerk_done))
        t.start()
    for c in range(cones_count):          # wait for clerks
        down(clerk_done)

    mess = "customer %d: served and going to pay" % (id)
    show(mess)
    walk_to_cachier()
    down(LINE.lock)                       # enter critical region
    num = LINE.number
    LINE.number += 1
    up(LINE.lock)                         # leave critical region
    up(LINE.requested)                    # signal cachier
    down(LINE.customers[num])             # wait in line

def check_out(i):
    pass

def cachier():

    global LINE

    for i in range(10):
        down(LINE.requested)              # wait for pay request
        mess = "cachier: customer %d paid" % i
        show(mess)
        check_out(i)
        up(LINE.customers[i])             # signal customer

def main():

    # set up cones' data

    cones = [random.choice(range(1,5)) for i in range(10)]
    tot_cones = sum(cones)
    mess = "main: %d cones %s" % (tot_cones, str(cones))
    show(mess)

    # create threads

    cts = [threading.Thread(target=customer, args=(i,n))
           for i,n in enumerate(cones)]
    ct = threading.Thread(target=cachier)
    mt = threading.Thread(target=manager, args=(tot_cones,))


    ct.start()                        # start threads
    mt.start()
    for t in cts:
        t.start()

    for t in cts:     # wait for threads to finish
        ct.join()

    mt.join()
    ct.join()
```

```
if __name__ == '__main__':
    main()
```