```python
# -*- coding:iso-latin-1 -*-

import random                   # choice
import threading                # Semaphore Thread
import time                     # sleep
import os                       # write

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

def down(sem):
    sem.acquire()

def up(sem):
    sem.release()

acquire = wait = down
release = signal = up

class Inspection():

    # Questa classe, di cui viene creata una sola istanza, 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, 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 è implicita nel
    # fatto che c’è una coda (gestita con il vettore di semafori
    # CUSTOMERS).

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

```python
# manager   ----------------------------------------

def do_inspection(inspected_count):
    mess = "manager: inspection %d" % inspected_count
    show(mess)
    return random.choice(range(approved_rate + 1)) > 0

def manager(tot_cones, approved_rate, inspection):

    approved_count = 0
    inspected_count = 0
    while approved_count < tot_cones:

        wait(inspection.requested)
        inspection.passed = do_inspection(inspected_count)
        inspected_count += 1
        if inspection.passed:
            approved_count += 1
        signal(inspection.finished)

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

# clerk   ----------------------------------------

def make_cone(clerk, cone, customer):
    mess = "clerk %2d: making cone %d for customer %d" % (
        clerk, cone, customer)
    show(mess)

def log_inspection(clerk, cone, customer, passed):
    mess = "clerk %2d: cone %d for customer %d %s" % (
        clerk, cone, customer,
        ["REJECTED", "passed"][passed])
    show(mess)

def clerk(id, customer, cone, clerk_done, inspection):

    passed = False
    while not passed:
        make_cone(id, cone, customer)
        acquire(inspection.lock)       # enter critical region
        signal(inspection.requested)   # ask for inspection
        wait(inspection.finished)      # leave office
        passed = inspection.passed
        log_inspection(id, cone, customer, passed)
        release(inspection.lock)       # leave critical region
    signal(clerk_done)                 # signal customer
```

```python
# customer   ----------------------------------------

def browse_flavours(id, cones_count):
    mess = "customer %d: asking for %d cones" % (
        id, cones_count)
    show(mess)

def walk_to_cachier(customer_id):
    mess = "customer %d: served and going to cachier" % (
        customer_id)
    show(mess)

CLERK_COUNT = 0                 # just for logging

def customer(id, cones_count, line, inspection):

    # Questo semaforo viene passato a ciascun clerk al momento della
    # creazione del thread e ciascun clerk fa un 'up' quando ha
    # finito, quindi customer può fare altrettanti 'down' per
    # aspettare che tutti abbiano finito.

    clerk_done = threading.Semaphore(0)

    browse_flavours(id, cones_count)

    global CLERK_COUNT
    for c in range(cones_count):          # "create" clerk
        t = threading.Thread(target=clerk,
                             args=(CLERK_COUNT + 1,
                                   id, c, clerk_done, inspection))
        t.start()
        CLERK_COUNT += 1
    for c in range(cones_count):          # wait for clerks
        wait(clerk_done)

    walk_to_cachier(id)
    acquire(line.lock)          # enter critical region
    num = line.number           # get ticket number
    line.number += 1
    release(line.lock)          # leave critical region
    signal(line.requested)      # wake up cachier
    wait(line.customers[num])   # wait in line

# cachier   ----------------------------------------

def check_out(i):
    mess = "cachier: customer %d paid" % i
    show(mess)

def cachier(customers_count, line):

    for i in range(customers_count):
        wait(line.requested)        # for customers to show up
        check_out(i)
        signal(line.customers[i])
```

```
# main   ----------------------------------------

def main(customers_count,
         max_cones_per_customer,
         approved_rate):

    # set up cones' data

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

    inspection = Inspection()
    line = Line(customers_count)

    # create threads

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

    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__':

    import sys
    args = sys.argv[1:]
    argc = len(args)

    customers_count = 10
    max_cones_per_customer = 5
    approved_rate = 9

    if (argc > 0): customers_count = int(args[0])
    if (argc > 1): max_cones_per_customer = int(args[1])
    if (argc > 2): approved_rate = int(args[2])

    main(customers_count,
         max_cones_per_customer,
         approved_rate)
```