```
#!/usr/bin/env python

# simulates NMachines machines, plus a queue of jobs waiting to use them

# usage:  python MMk.py NMachines ArvRate SrvRate MaxSimtime

from SimPy.Simulation import *
from random import Random,expovariate

# globals
class G:
   Rnd = Random(12345)

class MachineClass(Process):
   SrvRate = None  # reciprocal of mean service time
   Busy = []  # busy machines
   Idle = []  # idle machines
   Queue = []  # queue for the machines
   NDone = 0  # number of jobs done so far
   TotWait = 0.0  # total wait time of all jobs done so far, including
                  # both queuing and service times
   def __init__(self):
      Process.__init__(self)
      MachineClass.Idle.append(self)  # starts idle
   def Run(self):
      while 1:
         # "sleep" until this machine awakened
         yield passivate,self
         MachineClass.Idle.remove(self)
         MachineClass.Busy.append(self)
         # take jobs from the queue as long as there are some there
         while MachineClass.Queue != []:
            # get the job
            J = MachineClass.Queue.pop(0)
            # do the work
            yield hold,self,G.Rnd.expovariate(MachineClass.SrvRate)
            # bookkeeping
            MachineClass.NDone += 1
            MachineClass.TotWait += now() - J.ArrivalTime
         MachineClass.Busy.remove(self)
         MachineClass.Idle.append(self)

class JobClass:
   def __init__(self):
      self.ArrivalTime = now()

class ArrivalClass(Process):
   ArvRate = None
   def __init__(self):
      Process.__init__(self)
   def Run(self):
      while 1:
         # wait for arrival of next job
         yield hold,self,G.Rnd.expovariate(ArrivalClass.ArvRate)
         J = JobClass()
         MachineClass.Queue.append(J)
         # any machine ready?
         if MachineClass.Idle != []:
            reactivate(MachineClass.Idle[0])

def main():
   NMachines = int(sys.argv[1])
   ArrivalClass.ArvRate = float(sys.argv[2])
   MachineClass.SrvRate = float(sys.argv[3])
   initialize()
   for I in range(NMachines):
      M = MachineClass()
      activate(M,M.Run())
   A = ArrivalClass()
   activate(A,A.Run())
   MaxSimtime = float(sys.argv[4])
   simulate(until=MaxSimtime)
   print MachineClass.TotWait/MachineClass.NDone

if __name__ == '__main__': main()



# simulates one cell in a cellular phone network; a major highway runs
# through it, so handoff calls come in at random times but stay active
# for a constant time, the time it takes to drive through the cell (in
# this simpler model, we assume that a handoff call lasts the entire
# time the car is in the cell); calls also originate at random times
# from local people, who stay in the cell; for them, calls last a random
# time; a call is dropped, not queued, if a channel is not available

# usage:

# python Cell.py HRate DrvTime LRate LDurRate NChn NRsrvd MaxSimTime

# where:

#    HRate = rate of arrivals of handoff calls (reciprocal of mean time
#            between arrivals)
#    DrvTime = drive-through time for the cell
#    LRate = rate of creations of local calls (reciprocal of mean time
#            between creations)
#    LDurRate = reciprocal of mean duration of local calls
#    NChn = number of channels
#    NRsrvd = number of channels reserved for handoff calls
#    MaxSimtime = amount of time to simulate

import sys,random

from SimPy.Simulation import *

class Globals:
   Rnd = random.Random(12345)

class Cell:
   NChn = None
   NRsrvd = None
   FreeChannels = None
   NNoLocalsAllowedPeriods = 0
   TimeNoLocalsAllowed = 0.0
   LatestStartNoLocalsAllowed = None
   def GrabAChannel():
      Cell.FreeChannels -= 1  # grab the channel
      if Cell.FreeChannels == Cell.NRsrvd:
         Cell.LatestStartNoLocalsAllowed = now()
   GrabAChannel = staticmethod(GrabAChannel)
   def ReleaseAChannel():
      Cell.FreeChannels += 1  # release the channel
      if Cell.FreeChannels == Cell.NRsrvd+1:
         Cell.NNoLocalsAllowedPeriods += 1
         Cell.TimeNoLocalsAllowed += now() - Cell.LatestStartNoLocalsAllowed
         Cell.LatestStartNoLocalsAllowed = None
   ReleaseAChannel = staticmethod(ReleaseAChannel)

class Arrivals(Process):
   NArrv = {'handoff':0,'local':0}  # numbers of calls arrived so far
   def __init__(self,Type,Arr,Dur):
      Process.__init__(self)
      self.Type = Type
      self.Arr = Arr
      self.Dur = Dur
   def Run(self):
      while 1:
```

```
         TimeToNextArrival = Globals.Rnd.expovariate(self.Arr)
         yield hold,self,TimeToNextArrival
         Arrivals.NArrv[self.Type] += 1
         C = Call(self.Type,self.Dur)
         activate(C,C.Run())

class Call(Process):
   NRej = {'handoff':0,'local':0}  # numbers of calls rejected so far
   def __init__(self,Type,Dur):
      Process.__init__(self)
      self.Type = Type
      self.Dur = Dur
   def Run(self):  # simulates one call
      if self.Type == 'handoff' and Cell.FreeChannels == 0 or \
         self.Type == 'local' and Cell.FreeChannels <= Cell.NRsrvd:
            Call.NRej[self.Type] += 1
            return
      Cell.GrabAChannel()
      if self.Type == 'handoff':
         CallTime = self.Dur
      else:  # 'local'
         CallTime = Globals.Rnd.expovariate(self.Dur)
      yield hold,self,CallTime  # the call runs its course
      Cell.ReleaseAChannel()

def main():
   HRate = float(sys.argv[1])
   DrvTime = float(sys.argv[2])
   LRate = float(sys.argv[3])
   LDurRate = float(sys.argv[4])
   Cell.NChn = int(sys.argv[5])
   Cell.FreeChannels = Cell.NChn
   Cell.NRsrvd = int(sys.argv[6])
   initialize()
   HA = Arrivals('handoff',HRate,DrvTime)
   activate(HA,HA.Run())
   LCr = Arrivals('local',LRate,LDurRate)
   activate(LCr,LCr.Run())
   MaxSimtime = float(sys.argv[7])
   simulate(until=MaxSimtime)
   print 'percentage of rejected handoff calls:', \
      Call.NRej['handoff']/float(Arrivals.NArrv['handoff'])
   print 'percentage of rejected local calls:', \
      Call.NRej['local']/float(Arrivals.NArrv['local'])
   print 'mean banned period for locals', \
      Cell.TimeNoLocalsAllowed/float(Cell.NNoLocalsAllowedPeriods)

if __name__ == '__main__': main()



#!/usr/bin/env python

# MachRep4.py

# SimPy example:  R machines, which sometimes break down.  Up time is
# exponentially distributed with rate UpRate, and repair time is
# exponentially distributed with rate RepairRate.  The repairperson is
# summoned when fewer than K of the machines are up, and reaches the
# site after a negligible amount of time.  He keeps repairing machines
# until there are none that need it, then leaves.

# usage:  python MachRep4.py R UpRate RepairRate K MaxSimTime

from SimPy.Simulation import *
from random import Random,expovariate

class G:  # globals
   Rnd = Random(12345)
   RepairPerson = Resource(1)
   RepairPersonOnSite = False
   RPArrive = SimEvent()

class MachineClass(Process):
   MachineList = []  # list of all objects of this class
   UpRate = None  # reciprocal of mean up time
   RepairRate = None  # reciprocal of mean repair time
   R = None  # number of machines
   K = None  # threshold for summoning the repairperson
   TotalUpTime = 0.0  # total up time for all machines
   NextID = 0  # next available ID number for MachineClass objects
   NUp = 0  # number of machines currently up
   # create an event to signal arrival of repairperson
   def __init__(self):
      Process.__init__(self)
      self.StartUpTime = None  # time the current up period started
      self.ID = MachineClass.NextID   # ID for this MachineClass object
      MachineClass.NextID += 1
      MachineClass.MachineList.append(self)
      MachineClass.NUp += 1  # start in up mode
   def Run(self):
      from SimPy.Simulation import _e
      while 1:
         self.StartUpTime = now()
         yield hold,self,G.Rnd.expovariate(MachineClass.UpRate)
         MachineClass.TotalUpTime += now() - self.StartUpTime
         MachineClass.NUp -= 1
         # if the repairperson is already onsite, just request him;
         # otherwise, check whether fewer than K machines are up
         if not G.RepairPersonOnSite:
            if MachineClass.NUp < MachineClass.K:
                  G.RPArrive.signal()
                  G.RepairPersonOnSite = True
            else: yield waitevent,self,G.RPArrive
         yield request,self,G.RepairPerson
         yield hold,self,G.Rnd.expovariate(MachineClass.RepairRate)
         MachineClass.NUp += 1
         # if no more machines waiting for repair, dismiss repairperson
         if G.RepairPerson.waitQ == []:
            G.RepairPersonOnSite = False
         yield release,self,G.RepairPerson

def main():
   initialize()
   MachineClass.R = int(sys.argv[1])
   MachineClass.UpRate = float(sys.argv[2])
   MachineClass.RepairRate = float(sys.argv[3])
   MachineClass.K = int(sys.argv[4])
   for I in range(MachineClass.R):
      M = MachineClass()
      activate(M,M.Run())
   MaxSimtime = float(sys.argv[5])
   simulate(until=MaxSimtime)
   print 'proportion of up time was',  \
      MachineClass.TotalUpTime/(MachineClass.R*MachineClass.MaxSimtime)

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