

****

实验报告

内容（名称）：队列模型(M/M/1)设计与仿真

|  |  |
| --- | --- |
| 院（系）名称 | 计算机学院 |
| 专业名称 | 计算机科学与技术 |
| 指导教师 | 宋晓 |
| 学号 | 17373126 |
| 姓名 | 刘萱 |

2019年10月

**队列模型实验报告**

1. **实验目的**

应用M/M/1队列编程思想，模拟一台ATM自动取款机的排队过程，熟悉离散事件推进方式、队列建立和提取方式。

1. **数学模型**
2. 事件调度法，核心是创建一根事件轴和一支队列。先判定事件轴是否忙碌，是就根据时间先后顺序让顾客进入队列，否则推进事件。
3. 顾客到达按泊松分布生成；服务时间按指数分布生成。
4. 队列模型本身可以看作是一种“加入-离开”过程，与传染病模型的模式比较相像，请同学们给出自己对两类模型的比较分析。

两类模型的输入都是离散变量，概率模型较为相似；不同点在于传染病模型还受到治愈率等条件的影响。

1. 简述事件调度法、活动扫描法和进程交互法的异同。

事件调度法是面向事件建立仿真模型，记录事件发生的过程，处理每个事件发生时系统状态变化的结果；活动扫描法是面向活动建模，它记录每个活动开始与终止的时间，从而记录实体从一种状态变为另一种状态的过程；进程交互法是面向进程建模，它记录每个进程推进的过程，由于各进程是并行进行的，为了便于计算机处理，进程交互法采用交叉推进的方法，推进每个进程，最终完成全部进程的推进，即完成系统的全部运行过程。

所有策略均提供主动成分及被动成分，每种成分均能接受其他成分的作用。

在事件调度法中，用户要对所定义的全部事件进行建模，条件的测试只能在事件处理子例程中进行；活动扫描法设置了一个条件子例程专用于条件测试，还设置一个活动扫描模块，该模块对所有定义的活动进行建模。

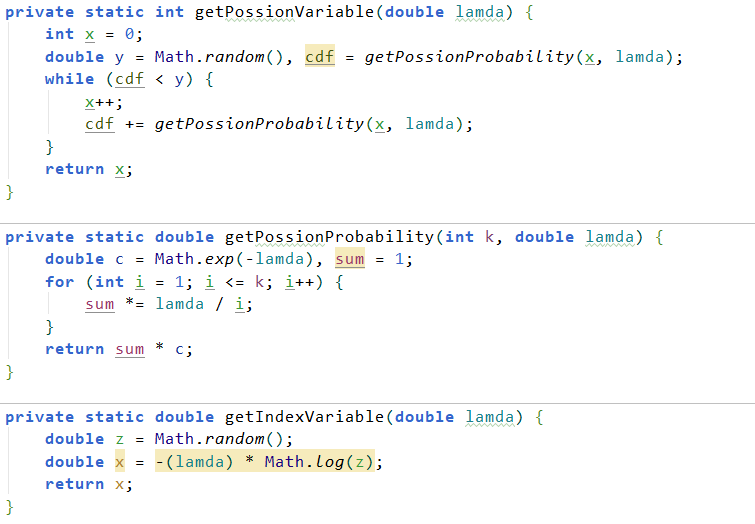
事件调度法由定时模块按下一最早发生时间选择事件记录，并转向该事件处理子程序执行。活动扫描法按递减优先数的顺序对全部活动扫描，只有满足测试条件为真，仿真事件小于等于系统仿真钟的活动才能被执行。进程交互法按递减优先数的顺序对当前事件表的全部记录进行扫描，根据该事件在其进程中的指针进行条件判断。

1. **编程实现与调试过程**

当客户到达时，需要记录客户的到达时间，如果ATM机处于空闲状态，就对客户进行服务；如果ATM机处于忙碌状态，则客户进入排队队列。当客户完成业务后，记录该客户的离开时间，同时可以通过计算得到他在队列中的等待时间。

由于每个客户的到达间隔时间符合泊松分布，服务时间符合指数分布，所以在这个客户到达时，就可以得到他的到达时间和所需要的服务时间。

这两个时间的生成函数如下：



完整代码见附录。

下面给出程序运行的一个结果实例：

输入 （时间单位：min）：

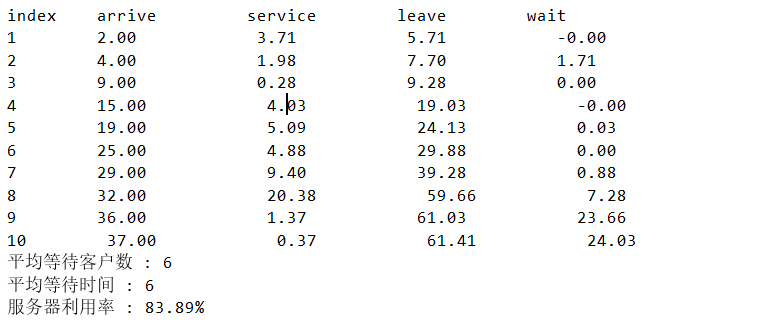
平均到达时间：3，

平局服务时间： 4，

顾客数目： 10，

队列最大长度 9

输出：



经过多次模拟，展示结果如下：

输入参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 平均到达时间(min) | 平均服务时间（min） | 服务人数 | 队列最大长度 |
| 3 | 4 | 100 | 30 |

**附录：**

public class Customer {

   private double arriveTime;

​

   private double serviceTime;

​

   private double leaveTime;

​

   private double waitTime;

​

​

   public void setArriveTime(double arriveTime) {

       this.arriveTime = arriveTime;

  }

​

   public void setServiceTime(double serviceTime) {

       this.serviceTime = serviceTime;

  }

​

   public void setLeaveTime(double leaveTime) {

       this.leaveTime = leaveTime;

       double time = this.leaveTime - this.arriveTime;

       this.setWaitTime(time - this.serviceTime);

  }

​

   public void setWaitTime(double waitTime) {

       this.waitTime = waitTime;

  }

​

   public double getArriveTime() {

       return arriveTime;

  }

​

   public double getServiceTime() {

       return serviceTime;

  }

​

   public double getLeaveTime() {

       return leaveTime;

  }

​

   public double getWaitTime() {

       return waitTime;

  }

}

​

public class Time {

   private double initial;

​

   public void initial() {

       this.initial = System.currentTimeMillis() / 60000;

  }

​

   public double getInitial() {

       return initial;

  }

}

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

​

public class Active {

   private  int getPossionVariable(double lamda) {

       int x = 0;

       double y = Math.random(), cdf = getPossionProbability(x, lamda);

       while (cdf < y) {

           x++;

           cdf += getPossionProbability(x, lamda);

      }

       return x;

  }

​

   private  double getPossionProbability(int k, double lamda) {

       double c = Math.exp(-lamda), sum = 1;

       for (int i = 1; i <= k; i++) {

           sum \*= lamda / i;

      }

       return sum \* c;

  }

​

   private  double getIndexVariable(double lamda) {

       double z = Math.random();

       double x = -(lamda) \* Math.log(z);

       return x;

  }

​

   public Active(double t1,double t2,int numCus,int maxNum) {

       Queue<Customer> queue = new LinkedList<>();

       Queue<Customer> finish = new LinkedList<>();

       ArrayList<Customer> list = new ArrayList<>();

       ArrayList<Integer> count = new ArrayList<>();

​

       Time time = new Time();

       time.initial();

       double nowTime = time.getInitial();

​

       for (int i = 0; i < numCus; i++) {

           Customer customer = new Customer();

           nowTime += getPossionVariable(t1);

           customer.setArriveTime(nowTime);

           customer.setServiceTime(getIndexVariable(t2));

           queue.add(customer);

      }

       nowTime = time.getInitial();

​

       while(!queue.isEmpty()){

           Customer customer = queue.peek();

           if(customer.getArriveTime() <= nowTime){

               customer.setLeaveTime(nowTime + customer.getServiceTime());

               nowTime += customer.getServiceTime();

               queue.poll();

               finish.add(customer);

               list.add(customer);

          }else {

               nowTime = customer.getArriveTime();

          }

      }

       nowTime = time.getInitial();

       double finalTime = list.get(list.size()-1).getLeaveTime();

       int i = 0;

       while(nowTime < finalTime){

           if((i < list.size()) && (list.get(i).getArriveTime() <= nowTime) && (queue.size() < maxNum)){

               queue.add(list.get(i));

               i++;

          }

           count.add(queue.size());

           nowTime++;

      }

       double waitTimeSum = 0;

       double serviveTimeSum = 0;

​

       while (!finish.isEmpty()) {

           Customer customer = finish.peek();

           if(customer.getWaitTime() > 0){

               waitTimeSum += customer.getWaitTime();

          }

           serviveTimeSum += customer.getServiceTime();

​

           finish.poll();

      }

       DecimalFormat dff = new DecimalFormat( "0 ");

       System.out.println(dff.format(Math.ceil(waitTimeSum / numCus)));

       int sum = 0;

       for(i=0;i<count.size();i++){

           sum += count.get(i);

      }

       System.out.println(dff.format(Math.ceil(sum / count.size())));

       DecimalFormat df = new DecimalFormat( "0.00% ");

       System.out.println(df.format(serviveTimeSum / (finalTime - time.getInitial())));

  }

}

import java.util.\*;

​

public class Main {

​

   public static void main(String[] args) {

       Scanner scanner = new Scanner(System.in);

       double t1 = scanner.nextDouble();

       double t2 = scanner.nextDouble();

       int numCus = scanner.nextInt();

       int maxNum = scanner.nextInt();

​

       for (int i = 0; i < 20; i++) {

           Active active = new Active(t1, t2, numCus, maxNum);

      }

  }

}