北京航空航天大學
BEIHANG UNIVERSITY

实验报告

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	17373240
姓名	赵婉如

2019 年 10 月

目录

一、 实验目的

应用$M/M/1$队列编程思想，模拟服务器处理顾客请求的过程，熟悉离散事件推进方式、队列建立和提取方式。通过纯数学和编程模拟实验来具体分析，向$M/M/1/K$队列与$M/M/N$队列进行扩展，并由此提出实际排队场景的优化策略。

二、数学模型引入

1. Kendall表示法

由David_George_Kendall提出，是用于描述排队系统的标准。

A / B / m / K / n / D

其中，

A 表示顾客到达流或顾客到达间隔时间的分布；

B 表示服务时间的分布

m 表示服务台数目

K 表示系统容量限制

n 表示顾客源数目

D 表示服务规则，如先到先服务 FCFS，后到先服务 LCFS 等。

约定：如略去后三项，即指$A/B/m/\infty/\infty/FCFS$的情形。本文主要探讨M/M/1的情况。

2. M/M/1排队模型（M/M/1 model）

一种单一服务器（single-server）的排队模型，可用作模拟不少系统的运作。

- 到达时间泊松过程（Poisson process）；
- 服务时间是指数分布（exponentially distributed）；
- 只有一部服务器（server），遵循先到先服务规则
- 队列长度无限制
- 可加入队列的人数为无限

约定符号表示如下：

- 顾客到达速率 λ
- 服务器服务速率 μ
- 服务被占用的平均概率 $\rho = \frac{\lambda}{\mu}$

3. 排队系统的运行指标

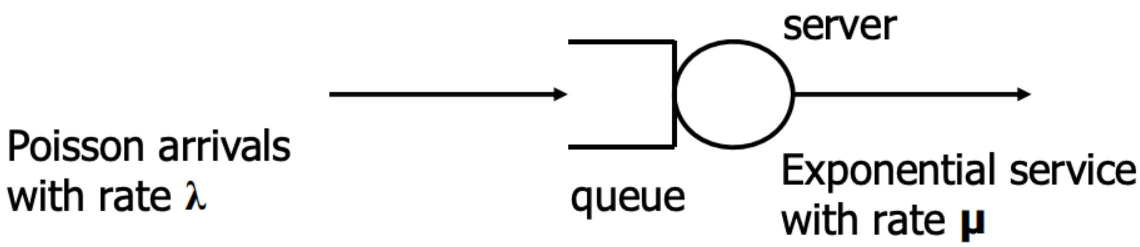
- 平均等待队列长度:指系统内等待服务的顾客数的数学期望。
- 平均逗留时间：顾客在系统内逗留时间（包括排队等待的时间和接受服务的时间）的数学期望。
- 平均等待时间：指一个顾客在排队系统中排队等待时间的数学期望。
- 服务器利用率：当前所有顾客服务的总时间与总时间跨度比值的数学期望。

三、数学分析

1. 单服务台模型

1. $M/M/1$

顾客的相继到达时间服从参数为λ的负指数分布，服务台个数为1，服务时间V服从参数为μ的负指数分布，系统空间无限，允许无限排队，这是一类简单的排队系统。



正如在概率论中所学过的，我们说随机变量 $\{N(t) = N(s+t) - N(s)\}$ 服从泊松分布。它的数学期望和方差分别是

$$E[N(t)] = \lambda t\text{；}\ \mathrm{Var}[N(t)] = \lambda t\text{。}$$

当输入过程是泊松流时，那么顾客相继到达的时间间隔 T 必服从指数分布。这是由于

$$P\{T > t\} = P\{[0,t)\text{ 内呼叫次数为零}\} = P_0(t) = e^{-\lambda t}$$

那么，以 $F(t)$ 表示 T 的分布函数，则有

$$P\{T \le t\} = F(t) = \begin{cases} 1 - e^{-\lambda t}, & t \ge 0 \\ 0, & t < 0 \end{cases}$$

而分布密度函数为

$$f(t) = \lambda e^{-\lambda t},\quad t > 0.$$

由此，可给出各测量数值的公式：

队列中平均顾客数:

$E[N] = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$

$E\left[N_q\right]=\frac{\rho^2}{1-\rho}$

队列中平均等待时间:

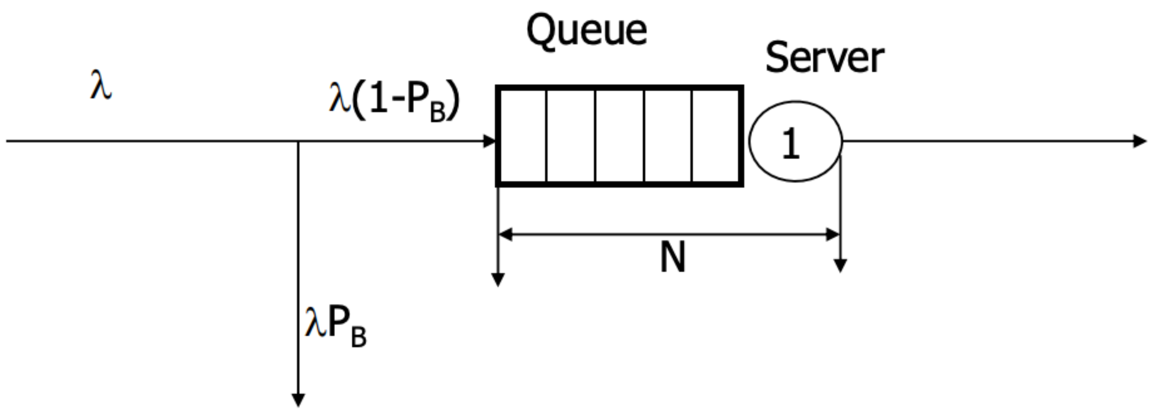
$E[T]=\frac{E[N]}{\lambda}=\frac{1}{\mu-\lambda}$

$E\left[T_q\right]=E[T]-E\left[T_s\right]=\frac{\lambda}{(\mu-\lambda)\mu}$

上述式子由利特尔法则（英语：Little's law）推出。利特尔法则排队理论，由约翰·利特尔在1954年提出。可用于一个稳定的、非占先式的系统中。其内容为：

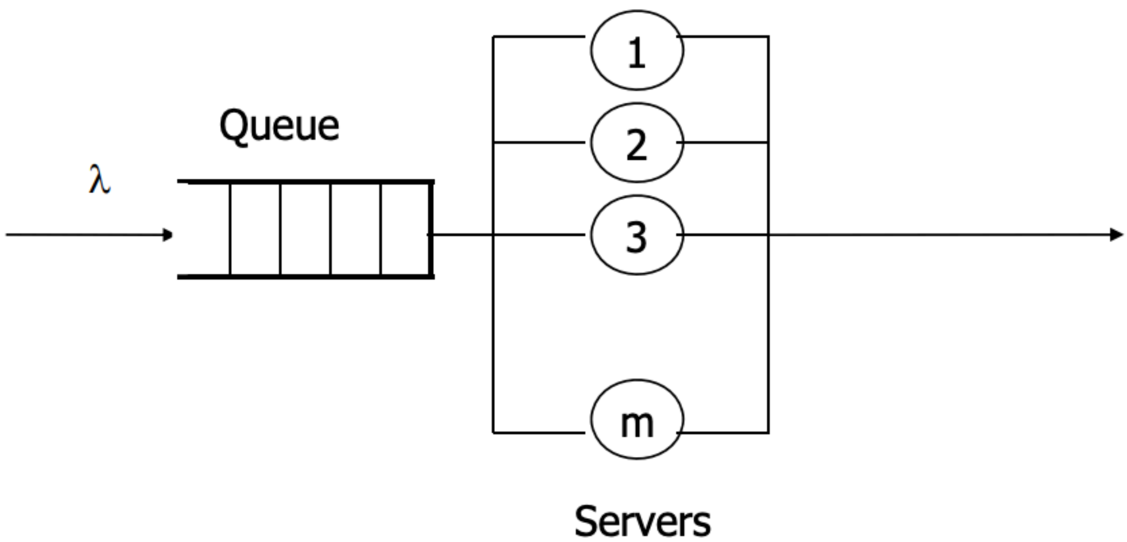
> 在一个稳定的系统中，长期的平均顾客人数（L），等于长期的有效抵达率（λ），乘以顾客在这个系统中平均的等待时间（W）用代数式表达为：
>
> $L=\lambda W$

2. $M/M/1/N$



2. 多服务台模型（M/M/s/∞）

1. $M/M/N$

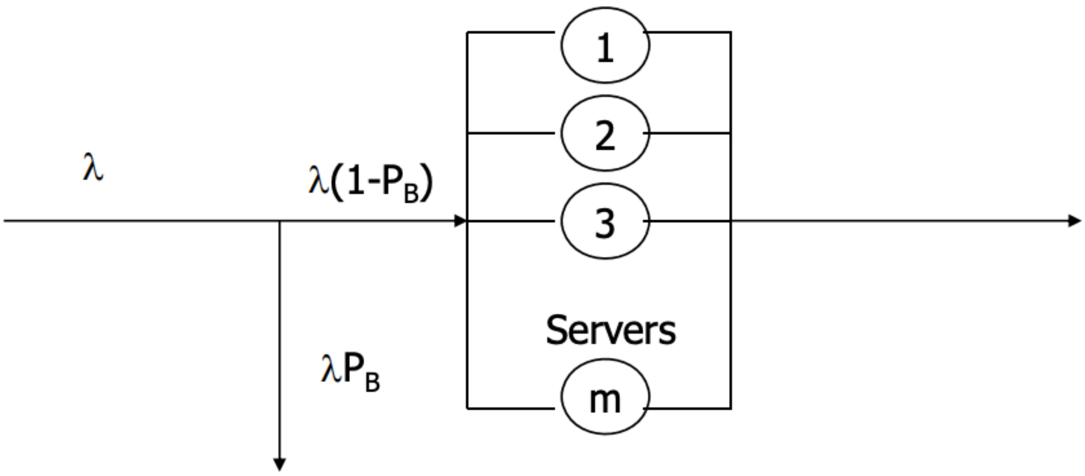


由Erlang等待公式，可以得到顾客到达系统时需要等待的概率

$$\mathrm{P_Q} = \sum_{n=m}^{\infty} P_n = \left[\frac{\rho^m m^m}{m!(1-\rho)}\right] p_0 = \frac{\frac{\rho^m m^m}{m!(1-\rho)}}{\left[\sum_{n=0}^{m-1}\frac{(m\rho)^n}{n!}+\frac{\rho^m m^m}{m!(1-\rho)}\right]}$$

系统总顾客数为N的概率 $\mathbb{E}[N] = \frac{\lambda}{\mu} + \pi_0 \frac{\rho(c\rho)^c}{(1-\rho)^2 c!}$

2. $M/M/1$



四、编程模拟

1. $M/M/1$

```
# 输入参数
total_time = int(input("输入模拟时间 (小时): "))
IAT_rate = int(input("输入顾客到达速率 (/小时): "))
ST_rate = int(input("输入服务器服务速率 (/小时): "))
# rho = IAT_rate / ST_rate
```

```
# 初始化参数
qu = queue.Queue() # 等待队列
curr_process = None # 当前服务的进程(顾客)
IAT = [] # 顾客到达时间间隔
ST = [] # 服务时间
AT = [] # 顾客到达
wait_time = [] # 等待时间
server_busy = False #服务器是否在忙
# 记录每秒的运行指标, 方便画图
list_wait = [] # 平均等待时长
list_delay = [] # 平均逗留时长
list_queue_wait = [] # 队列中平均等待顾客数
list_server_busy = [] # 服务器利用率
```

当输入过程是泊松流的时候，顾客相继到达的时间间隔必服从指数分布。 据此生成满足概率分布的随机数，并初始化所有顾客”到达—离开“的时间：

```
# 泊松分布取随机数 总共需要服务的人数
num_processes = int(np.random.poisson(IAT_rate) * total_time)
# 已经服务的人数
num_processes_served = 0

# 计算顾客到达时间间隔IAT (Inter-Arrival-Times)
for i in range(num_processes):
    temp = np.random.exponential(1 / IAT_rate) * 60 * 60
    if i == 0:
        IAT.append(0)
    else:
        IAT.append(int(temp - temp % 1))

# 计算服务器服务时长ST (Service-Times)
while not len(ST) == num_processes:
    temp = np.random.exponential(1 / ST_rate) * 60 * 60
    if not int(temp - temp % 1) < 1:
        ST.append(int(temp - temp % 1))
```

```
# 保存ST值
ST_copy = copy.deepcopy(ST)

# 计算顾客到达时间AT (Arrival-Times) 并将等待时间(Waiting-Times)初始化为0
for i in range(num_processes):
    if i == 0:
        AT.append(0)
    else:
        AT.append(AT[i - 1] + IAT[i])
    wait_time.append(0)
```

仿真核心过程：采用**活动扫描法**。遍历整个时间轴，可以实现对条件的处理。

```
# 仿真模拟 M/M/1
# (i表示当前时间)
sum_queue_wait=0
sum_server_busy=0
for i in range(total_time * 60 * 60):
    print("当前第{}秒:\n".format(i))
    if server_busy: # 服务器正忙
        print("顾客{}正在接受服务\n".format(curr_process))
        print("当前已服务人数:{}\n".format(num_processes_served))
        for item in list(qu.queue):
            wait_time[item] = wait_time[item] + 1
        ST[curr_process] = ST[curr_process] - 1
        if ST[curr_process] == 0: # 当前顾客服务完毕
```

```
            server_busy = False
            num_processes_served = num_processes_served + 1
            print("顾客{}已服务完毕\n".format(curr_process))

    for j in range(num_processes): # 顾客在当前时刻进入等待队列
        if i == AT[j]:
            qu.put(j)
            print("顾客{}进入等待队列，当前等待人数：{}\n".format(j,qu.qsize()))

    if not server_busy and not qu.empty():# 当前服务器闲置，开始服务下一位顾客
        curr_process = qu.get()
        print("顾客{}开始接受服务\n".format(curr_process))
        server_busy = True

    if not server_busy:
        print("当前服务器闲置\n")

    #统计指标
    sum_queue_wait += qu.qsize()
    list_queue_wait.append(sum_queue_wait / (i+1))

    sum_server_busy += server_busy
    list_server_busy.append(sum_server_busy/ (i+1))

    sum_wait = 0
    sum_delay = 0

    for i in range(num_processes_served):
        sum_wait = sum_wait + wait_time[i]
        sum_delay = sum_delay + wait_time[i] + ST_copy[i]

    if num_processes_served == 0:
        list_wait.append(0)
        list_delay.append(0)
    else:
        list_wait.append(sum_wait / (num_processes_served * 60 * 60))
        list_delay.append(sum_delay / (num_processes_served * 60 * 60))
```

绘制衡量指标随仿真时间变化的图像:

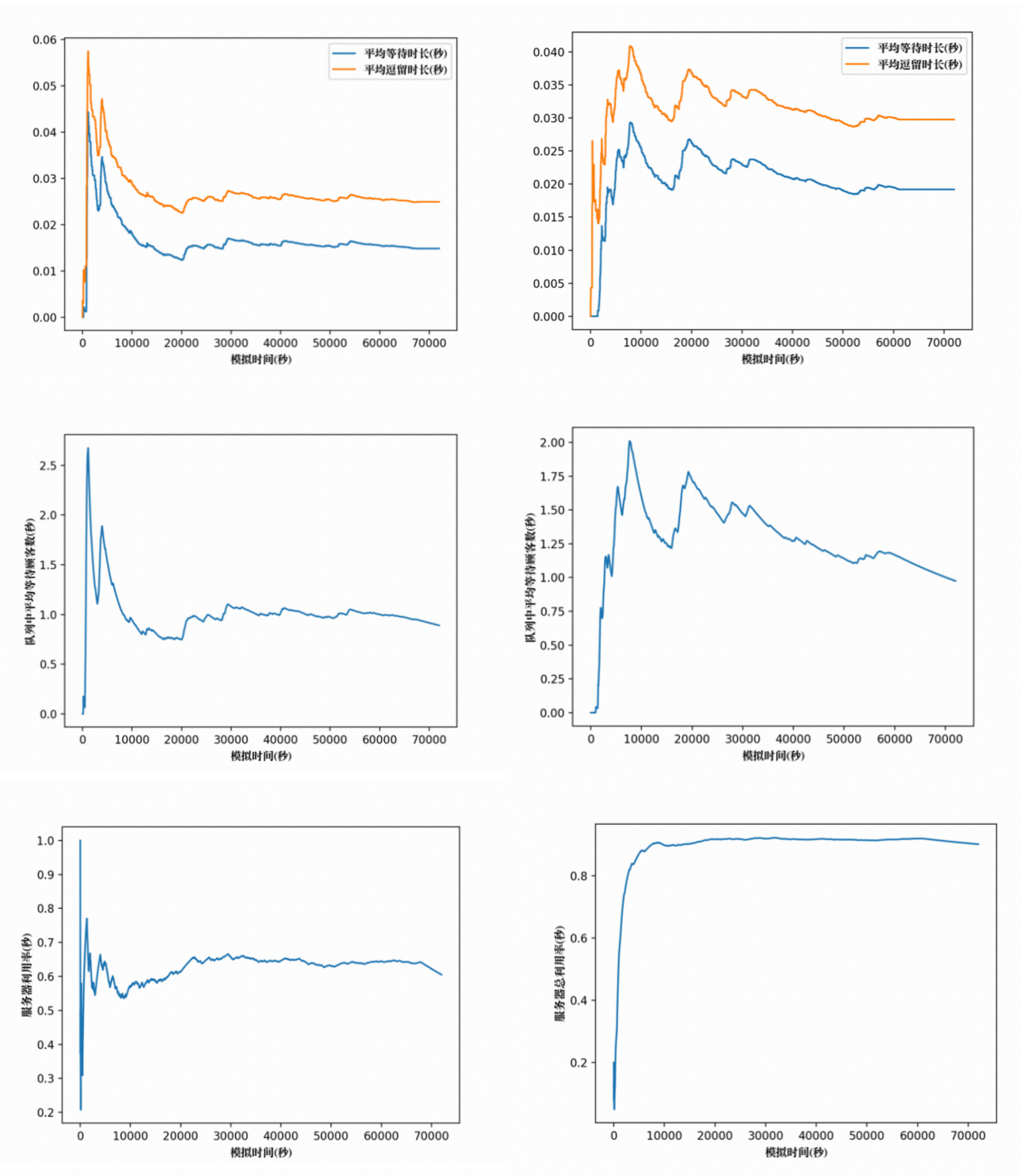
```
plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_wait,label=u"平均等待时长(秒)")
plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_delay,label=u"平均逗留时长(秒)")
#plt.legend(FontProperties=font)
plt.legend(prop=font)
plt.xlabel(u"模拟时间(秒)",FontProperties=font)
#plt.ylabel()
```

```
plt.show()

#plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_delay)
#plt.ylabel(u"平均逗留时长",FontProperties=font)
#plt.show()

plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_queue_wait)
plt.ylabel(u"队列中平均等待顾客数(秒)",FontProperties=font)
plt.xlabel(u"模拟时间(秒)",FontProperties=font)
plt.show()

plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_server_busy)
plt.ylabel(u"服务器利用率(秒)",FontProperties=font)
plt.xlabel(u"模拟时间(秒)",FontProperties=font)
plt.show()
```

程序输入参数值与运行效果如下：

```
输入模拟时间 (小时): 20
输入顾客到达速率 (/小时): 60
输入服务器服务速率 (/小时): 100
当前第0秒:

顾客0进入等待队列，当前等待人数： 1

顾客1进入等待队列，当前等待人数： 2

顾客0开始接受服务

当前第1秒:

顾客0正在接受服务

当前已服务人数:0

...

当前第25秒:

顾客0正在接受服务

当前已服务人数:0

顾客0已服务完毕

顾客1开始接受服务

当前第26秒:
```

```
顾客1正在接受服务

当前已服务人数:1

...

当前第41秒:

顾客1正在接受服务

当前已服务人数:1

顾客2进入等待队列, 当前等待人数: 1

...
```



M/M/1模型运行结果(左)与M/M/N模型运行结果(右)对比
输入参数:仿真时间20h,顾客到达速率60/小时,服务器服务速率100/小时,(服务器个数5)

2. $M/M/1/n$

对上述代码做如下增改:

```
#输入部分
queue_limit=int(input("输入队列最大长度 (人): "))
```

```python
#模拟部分
for i in range(total_time * 60 * 60):
      for j in range(num_processes):
        if i == AT[j] and qu.qsize()<queue_limit:
            qu.put(j)
            print("顾客{}进入等待队列，当前等待人数：{}\n".format(j,qu.qsize()))
      if qu.qsize()==queue_limit:
        print("当前等待队列已满\n")
```

3. $M/M/N$

对上述代码做如下增改：

```python
# 输入
server_num = int(input("输入服务器数量： "))
# 初始化
server_busy = 0
# 模拟
for i in range(total_time * 60 * 60):
    print("当前第{}秒:\n".format(i))
    if server_busy > 0:
        print("当前已经服务到第{}号顾客\n".format(curr_process))
        print("当前已结束服务的顾客数:{}\n".format(num_processes_served))
        for item in list(qu.queue):
            wait_time[item] = wait_time[item] + 1
        ST[curr_process] = ST[curr_process] - 1
        if ST[curr_process] == 0:
            server_busy -= 1
            num_processes_served = num_processes_served + 1
            print("顾客{}已服务完毕\n".format(curr_process))

    for j in range(num_processes):
        if i == AT[j]:
            qu.put(j)
            print("顾客{}进入等待队列，当前等待人数：{}\n".format(j,qu.qsize()))

    if server_busy < server_num and not qu.empty():
        curr_process = qu.get()
        print("顾客{}开始接受服务\n".format(curr_process))
        server_busy += 1

    if server_busy == 0:
        print("当前所有服务器闲置\n")

    #统计指标
    sum_queue_wait += qu.qsize()
```

```
    list_queue_wait.append(sum_queue_wait / (i+1))

    sum_server_busy += server_busy
    list_server_busy.append(sum_server_busy/ (server_num*(i+1)))
```

三、模型分析

1. 模型核心

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。

2. 与传染病模型的比较分析

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有相通之处。

传染病模型中，病人治愈成为健康人，健康人可被感染。

	传染病SIS模型	队列模型
λ	日接触率	顾客到达的速率
μ	治愈率	服务器处理速率
$\sigma=\lambda/\mu$	接触数(一个感染期内每个病人的有效接触的平均人数)	服务器被占用的平均概率
$\sigma=\lambda/\mu>1,\lambda>\mu,i\uparrow$	病人增多	供不应求
$\sigma=\lambda/\mu\le 1,\lambda\le\mu,i\downarrow$	病人减少	供大于求

3. 不同实现方法的比较：

模拟方法	异	同
事件调度法	每次寻找下一最早发生事件的发生时间，是一种"预定事件发生"的策略。但无法处理事件的发生不仅与事件有关，而且与其他条件有关，即只有满足某些条件时才会发生的情况。无法预定活动的开始和终止事件。	基础的方法
活动扫描法	设置一个活动扫描模块，不但扫描主动组件活泼发生的事件，还扫描活动发生的条件。能很好地解决事件调度法的缺陷：组件间的相互关系除了定义组件的活动之外，还包括对于"条件"的处理。	组件、变量的定义与事件调度法相同
进程交互法	采用进程描述系统，一个组件一旦进入进程，它将尽可能执行尽可能多的活动，遇到需要某些条件满足才能执行的活动则进程停止，与其他组件的进程实现交互。从用户的观念看，该策略更易于使用，但软件实现比上述两种方法要复杂得多。	既可预定时间，又可对条件求值，兼有上述两种方法的优点

四、 实际排队场景优化策略

以新北区一层食堂为例，给出初步的窗口优化策略

1. **拓扑结构**

   新北区一层食堂的拓扑结构如图1所示，左侧较大的区域为用餐区，右侧排队区通过间隔隔板与之隔开。学校师生在右侧饭菜窗口前排队。

2. **人流特点**

   饭菜窗口排列紧密，导致排队人流十分集中，且由于与面积较大的用餐区隔开，可供师生等候排队的面积较小，如图1阴影部分所示。导致等待队列的容量较小，饭菜窗口系统处理速度慢，所需买饭时间长。

3. **改进措施**

   根据上述对排队模型的分析提出改进措施，示意图如图2所示。具体方案如下：

   - 增大分隔板与饭菜窗口间的距离L,或取消分隔板，从而增大等待队列的人数$queue_limit$，由上述对$M/M/1/n$模型的分析可知，顾客的平均等待时间会大大减少。
   - 在食堂两侧位置增加饭菜窗口，从而增大服务窗口数量$server_num$，由上述对$M/M/N$模型的分析可知，顾客的平均等待时间会大大减少。

可达到的排队人流效果如图2阴影部分所示。

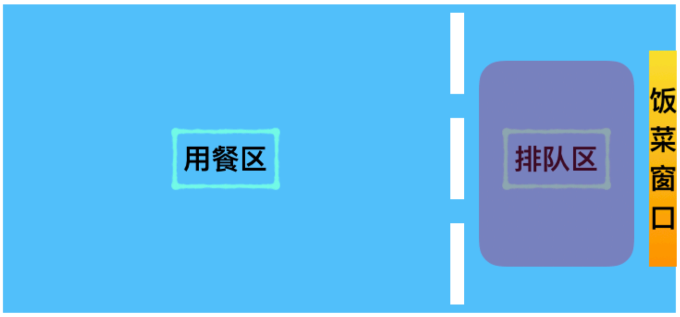


图1. 新北区食堂一层拓扑结构及人流特点

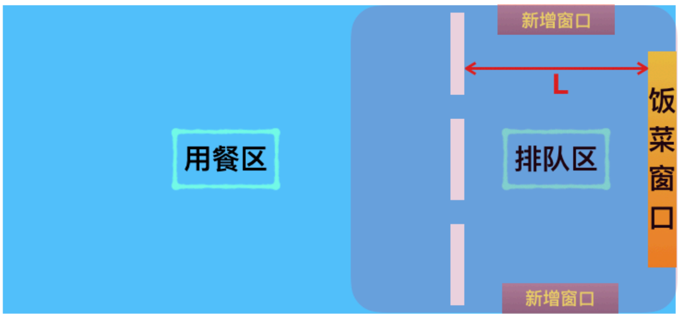


图2. 新北区食堂一层改进示意图

五、 参考文献

> [1] 《离散事件系统建模与仿真》肖田元、范文慧编著，电子工业出版社
>
> [2] 《计算机科学技术百科全书》 张效祥编著
>
> [3] [Basic Queueing Theory, Dr. János Sztrik, University of Debrecen, Faculty of Informatics]()
>
> [4] [等候理論 Queueing Theory]()

六、 附录

完整代码

```
import numpy as np
import queue
import copy
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties
font = FontProperties(fname='/Library/Fonts/Songti.ttc')

# 输入参数
total_time = int(input("输入模拟时间 (小时): "))
IAT_rate = int(input("输入顾客到达速率 (/小时): "))
```

```
ST_rate = int(input("输入服务器服务速率 (/小时): "))
rho = IAT_rate / ST_rate

# 初始化参数
qu = queue.Queue()
curr_process = None
IAT = []
ST = []
AT = []
wait_time = []
server_busy = False
list_wait = []
list_delay = []
list_queue_wait = []
list_server_busy = []

# 泊松分布取随机数 总共需要服务的人数
num_processes = int(np.random.poisson(IAT_rate) * total_time)
num_processes_served = 0

# 计算顾客到达时间间隔IAT (Inter-Arrival-Times)
for i in range(num_processes):
    temp = np.random.exponential(1 / IAT_rate) * 60 * 60
    if i == 0:
        IAT.append(0)
    else:
        IAT.append(int(temp - temp % 1))

# 计算服务器服务时长ST (Service-Times)
while not len(ST) == num_processes:
    temp = np.random.exponential(1 / ST_rate) * 60 * 60
    if not int(temp - temp % 1) < 1:
        ST.append(int(temp - temp % 1))

# 保存ST值
ST_copy = copy.deepcopy(ST)

# 计算顾客到达时间AT (Arrival-Times) 并将等待时间(Waiting-Times)初始化为0
for i in range(num_processes):
    if i == 0:
        AT.append(0)
    else:
        AT.append(AT[i - 1] + IAT[i])
    wait_time.append(0)

# 仿真模拟 M/M/1
# (i表示当前时间)
sum_queue_wait=0
sum_server_busy=0
```

```python
for i in range(total_time * 60 * 60):
    print("当前第{}秒:\n".format(i))
    if server_busy:
        print("顾客{}正在接受服务\n".format(curr_process))
        print("当前已服务人数:{}\n".format(num_processes_served))
        for item in list(qu.queue):
            wait_time[item] = wait_time[item] + 1
        ST[curr_process] = ST[curr_process] - 1
        if ST[curr_process] == 0:
            server_busy = False
            num_processes_served = num_processes_served + 1
            print("顾客{}已服务完毕\n".format(curr_process))

    for j in range(num_processes):
        if i == AT[j]:
            qu.put(j)
            print("顾客{}进入等待队列，当前等待人数：{}\n".format(j,qu.qsize()))

    if not server_busy and not qu.empty():
        curr_process = qu.get()
        print("顾客{}开始接受服务\n".format(curr_process))
        server_busy = True

    if not server_busy:
        print("当前服务器闲置\n")

    sum_queue_wait += qu.qsize()
    list_queue_wait.append(sum_queue_wait / (i+1))

    sum_server_busy += server_busy
    list_server_busy.append(sum_server_busy/ (i+1))

    sum_wait = 0
    sum_delay = 0

    for i in range(num_processes_served):
        sum_wait = sum_wait + wait_time[i]
        sum_delay = sum_delay + wait_time[i] + ST_copy[i]

    if num_processes_served == 0:
        list_wait.append(0)
        list_delay.append(0)
    else:
        list_wait.append(sum_wait / (num_processes_served * 60 * 60))
        list_delay.append(sum_delay / (num_processes_served * 60 * 60))


plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_wait,label=u"平均等待时长(秒)")
```

```
plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_delay,label=u"平均
逗留时长(秒)")
plt.legend(prop=font)
plt.xlabel(u"模拟时间(秒)",FontProperties=font)
#plt.ylabel()
plt.show()

plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_queue_wait)
plt.ylabel(u"队列中平均等待顾客数(秒)",FontProperties=font)
plt.xlabel(u"模拟时间(秒)",FontProperties=font)
plt.show()

plt.plot([i + 1 for i in range(total_time * 60 * 60)], list_server_busy)
plt.ylabel(u"服务器利用率(秒)",FontProperties=font)
plt.xlabel(u"模拟时间(秒)",FontProperties=font)
plt.show()
```