МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Исследование операций»

**Лабораторная работа №2.**

**Система массового обслуживания**

**многоканальная модель**

Выполнил:

студент группы ИВТАПбд-21

Кондратьев П. С.

Проверил:

Фролов В. А.

Ульяновск, 2018

**Система массового обслуживания**

Модели с n обслуживающими каналами. В подавляющем большинстве случаев на практике системы массового обслуживания являются многоканальными, и, следовательно, модели с n обслуживающими каналами (где n > 1) представляют несомненный интерес.

Процесс массового обслуживания, описываемый данной моделью, характеризуется интенсивностью входного потока λ, при этом параллельно может обслуживаться не более n клиентов (заявок). Средняя продолжительность обслуживания одной заявки равняется 1/μ. Входной и выходной потоки являются пуассоновскими. Режим функционирования того или иного обслуживающего канала не влияет на режим функционирования других обслуживающих каналов системы, причем длительность процедуры обслуживания каждым из каналов является случайной величиной, подчиненной экспоненциальному закону распределения. Конечная цель использования n параллельно включенных обслуживающих каналов заключается в повышении (по сравнению с одноканальной системой) скорости обслуживания требований за счет обслуживания одновременно n клиентов.

Граф состояний многоканальной системы массового обслуживания с отказами имеет вид, показанный на рис. 1.



Рис.1 Многоканальная СМО

**Примеры задач:**

Механическая мастерская завода с тремя постами (каналами) выполняет ремонт малой механизации. Поток неисправных механизмов, прибывающих в мастерскую,  - пуассоновский и имеет интенсивность λ=2,5 механизма в сутки, среднее время ремонта одного механизма распределено по показательному закону и равно t*об*=0,5 сут. Предположим, что другой мастерской на заводе нет, и, значит, очередь механизмов перед мастерской может расти практически неограниченно.

Требуется вычислить следующие предельные значения вероятностных характеристик системы:

- вероятность состояний системы;

- среднее число заявок в очереди на обслуживание;

- среднее число находящихся в системе заявок;

- среднюю продолжительность пребывания заявки в очереди;

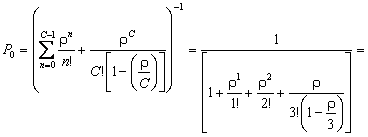
- среднюю продолжительность пребывания заявки в системе.

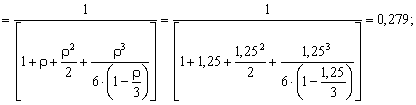
***Решение.***

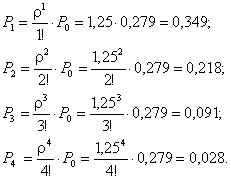
1) Определим параметр потока обслуживаний https://studfiles.net/html/2706/672/html_dai0KRngyu.rC7G/img-I_6BmW.png

2) Приведенная интенсивность потока заявок ρ=λ/μ=2,5/2,0=1,25, при этом λ/μ∙*с*=2,5/2∙3=0,41<1.      Поскольку λ/μ∙*с*<1, то очередь не растет безгранично и в системе наступает предельный стационарный режим работы.

3) Вычислим вероятности состояний системы:







4) Вероятность отсутствия очереди у мастерской:

Ротк≈*Р*0+*Р*1+*Р*2+*Р*3≈0,279+0,394+0,218+0,091=0,937.

5) Среднее число заявок в очереди на обслуживание:

https://studfiles.net/html/2706/672/html_dai0KRngyu.rC7G/img-krtjWj.png

6) Среднее число находящихся в системе заявок:

*Ls*=*Lq*+https://studfiles.net/html/2706/672/html_dai0KRngyu.rC7G/img-ZQ6Jej.png=0,111+1,25=1,361.

7) Средняя продолжительность пребывания механизма в очереди на обслуживание:

https://studfiles.net/html/2706/672/html_dai0KRngyu.rC7G/img-zlR79s.pngсуток.

8) Средняя продолжительность пребывания механизма в мастерской (в системе):

https://studfiles.net/html/2706/672/html_dai0KRngyu.rC7G/img-GTkbak.pngсуток.

**Исходный код**

#include <iostream>

#include <ctime>

#include <list>

//---------------------------------------------------------------------------

using namespace std;

//---------------------------------------------------------------------------

enum RequestType { firstType = 1, secType = 2 }; // Тип заявки.

//---------------------------------------------------------------------------

// Заявка: тип.

struct Request {

RequestType ReqType;

};

//---------------------------------------------------------------------------

const double BegInterval = 0; // Начало каждого из интервалов генерации и обработки.

const double EndGenerateFReq = 5; // Максимальное время генерации заявки 1-го типа.

const double UseFReq = 4; // Максимальное время обработки заявки 1-го типа.

const double UseSReq = 4; // Максимальное время обработки заявки 2-го типа.

const int MaxSReqPos = 4; // Не далее этого элемента заявка второго типа вернётся в очередь.

//---------------------------------------------------------------------------

// RandTime() генерирует случайное число.

double RandTime() {

double Val = (rand() \* 1.0) / RAND\_MAX;

return Val;

}

//---------------------------------------------------------------------------

// MinTime возвращается из 2-х значений.

double MinTime(double T1, double T2) {

return ((T1 > T2) ? T2 : T1);

}

//---------------------------------------------------------------------------

// GenAddFReq() генерирует время прихода следующей заявки 1-го типа.

double GenAddFReq() {

double Val = RandTime();

double Time = (EndGenerateFReq - BegInterval) \* Val + BegInterval;

return Time;

}

//---------------------------------------------------------------------------

// GenUseFReq() генерирует время обработки следующей заявки 1-го типа.

double GenUseFReq() {

double Val = RandTime();

double Time = (UseFReq - BegInterval) \* Val + BegInterval;

return Time;

}

//---------------------------------------------------------------------------

// GenUseSReq() генерирует время обработки заявки 2-го типа.

double GenUseSReq() {

double Val = RandTime();

double Time = (UseSReq - BegInterval) \* Val + BegInterval;

return Time;

}

//---------------------------------------------------------------------------

int main() {

srand(static\_cast<unsigned int>(time(NULL)));

list<RequestType> RequestQueue;

RequestType STypeReq = secType;

RequestType FTypeReq = firstType;

RequestQueue.push\_back(STypeReq);

unsigned int CountFReqs = 0; // Количество обработанных заявок 1-го типа для вывода статистики.

unsigned int CountUseFReqs = 0; // Количество обработанных заявок 1-го типа.

unsigned int Enter2QueueFReqs = 0; // Количество заявок 1-го типа вошедших в систему.

double Time = 0.0; // Время, минимум между поступлением новой заявки и обработкой текущей.

double TimeEnter = 0.0; // Время прихода заявки.

double TimeClearService = 0.0; // Время обработки заявки (освобождения ОА)

double NextEventTime = 0.0; // Время следующего события.

double TotalTime = 0.0; // Общее время моделирования.

list<RequestType>::iterator PredSReq = RequestQueue.begin(); // Этот итератор указывает на заявку,

// перед которой надо вставить заявку 2го типа.

list<RequestType>::iterator tmp = RequestQueue.begin();

unsigned long CountOfCalcs = 0; // Количество вычислений.

double MiddleLenQueue = 0.0; // Средняя длина после каждого вычисления.

unsigned long countSecReqs = 0; // Количество заявок 2-го типа.

unsigned long SumOfCalcs = 0; // Сумма всех измерений.

while (true) {

if (1000 == CountUseFReqs) break;

TimeEnter = GenAddFReq();

if (2 == RequestQueue.front()) TimeClearService = GenUseSReq();

else TimeClearService = GenUseFReq();

Time = MinTime(TimeEnter, TimeClearService);

//cout << "[Time]: " << Time << " | " << RequestQueue.front() << endl;

if (Time == TimeEnter) {

// Добавляем заявку первого типа.

RequestQueue.push\_back(FTypeReq);

CountFReqs++; // Увеличиваем счётчик заявок 1-го типа, для вывод статистики и выхода из цикла.

Enter2QueueFReqs++; // Увеличиваем общее количество заявок кошедших в систему.

TotalTime += Time;

continue;

}

else {

// Обрабатываем заявку.

if (secType == RequestQueue.front()) {

// Обрабатываем заявку 2-го типа.

countSecReqs++;

RequestQueue.pop\_front();

if (MaxSReqPos >= RequestQueue.size()) {

RequestQueue.push\_back(STypeReq);

continue;

}

else {

PredSReq = RequestQueue.begin();

for (int i = 0; i != 4; i++) PredSReq++;

RequestQueue.insert(PredSReq, STypeReq);

continue;

}

}

else {

// Обрабатываем заявку 1-го типа.

TotalTime += Time;

CountUseFReqs++;

RequestQueue.pop\_front();

SumOfCalcs += static\_cast<int>(RequestQueue.size());

CountOfCalcs++;

if ((0 == CountUseFReqs % 100) && (0 != CountUseFReqs)) {

MiddleLenQueue = (SumOfCalcs \* 1.0) / CountOfCalcs;

cout << "----------" << endl;

cout << "[Count of requests]: " << CountUseFReqs << endl;

cout << "Now length of queue: " << static\_cast<unsigned int>(RequestQueue.size()) << endl;

cout << "Now middle length of queue: " << MiddleLenQueue << endl;

cout << "[countSecReqs]: " << countSecReqs << endl;

}

}

}

}

cout << "----------" << endl << "Total modeling time: " << TotalTime << " ms" << endl;

cout << "1st. type requests enter: " << Enter2QueueFReqs << endl;

cout << "1st. type requests out: " << CountUseFReqs << endl;

cout << "[countSecReqs]: " << countSecReqs << endl;

cout << "Time: " << TimeEnter + TimeClearService << " s" << endl;

int i;

cin >> i;

return 0;

}