Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 4

Дисциплина: Моделирование

Тема: Моделирование системы с обслуживающим аппаратом

Вариант: 9

Студент: Платонова О. С.

Группа: ИУ7-75Б

Оценка (баллы): _____

Преподаватель: Рудаков И. В.

Москва, 2021 г.

Цель работы: моделирование системы, состоящей из генератора, очереди сообщений и обслуживающего аппарата. Генератор создает сообщения, распределенные по нормальному закону. Сообщения попадают в очередь и обрабатываются по закону Пуассона. Предусмотреть ввод параметров законов распределения пользователем.

Необходимо определить длину очереди, при которой не будет потерянных сообщений. Также требуется реализация двух принципов: событийного и итерационного. Шаг итерации задает пользователь.

Законы распределения

Законы распределения соответствуют законам 1ой лабораторной. Параметры вводит пользователь.

Равномерное распределение $\mathcal{U}(a, b)$

Функция распределения:

$$F_X(x) \equiv P(X \leq x) = \begin{cases} 0, & x < a \\ \dfrac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases}$$

Функция плотности:

$$f_X(x) = \begin{cases} \dfrac{1}{b-a}, & x \in [a, b] \\ 0, & x \notin [a, b] \end{cases}$$

Распределение Пуассона P(λ)

Функция распределения:

$$F_X(x) \equiv P(X = x) = \sum_{k=0}^{x} \frac{\lambda^k e^{-\lambda}}{k!}$$

Функция плотности:

$$f_X(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Итерационная модель

Время генерации и обслуживания сообщения задаются в соответствии с законами распределения. Далее итерационно (до тех пор, пока количество обслуженных сообщений не превышает общего количества) выполняется генерация сообщений или их обслуживание из очереди.

Если текущее время работы превышает время генерации, то следует сгенерировать новое сообщение.

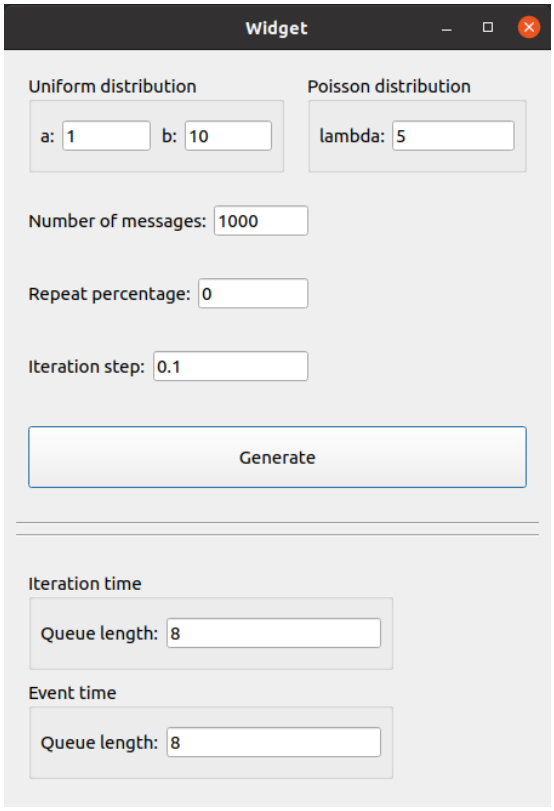
В случае, когда текущее время работы превышает время обработки, то выполняется обработка сообщения из очереди.

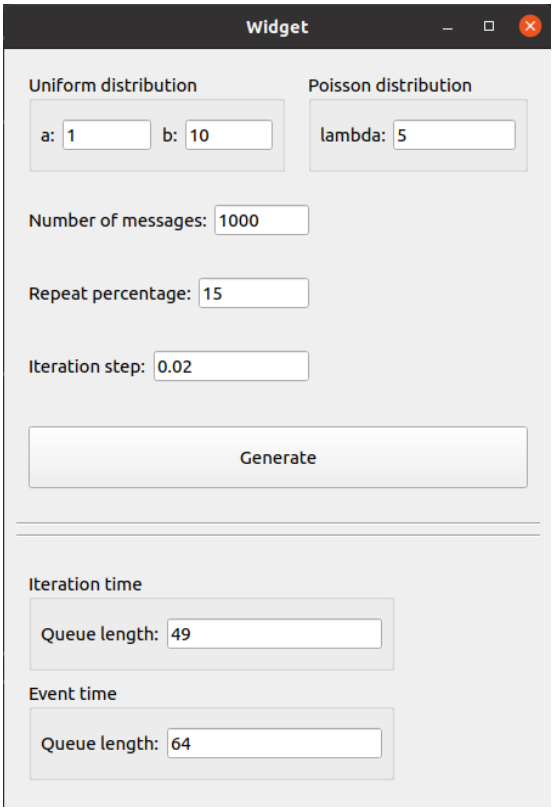
Происходит увеличение текущего времени на величину шага, заданного пользователем. Следует также учитывать процент повторных сообщений. Время обработки соответствует случайному числу, распределенному по закону Пуассона; время генерации – по равномерному закону.

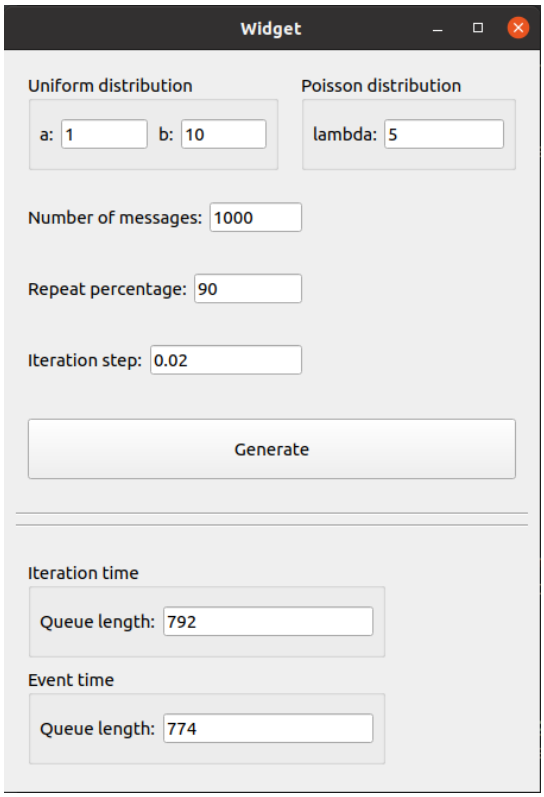
Событийная модель

Время генерации и обслуживания также, как и в итерационной модели, зависит от законов распределения. Выполняется цикл (пока количество обработанных сообщений меньше общего заданного количества), в котором определяется наиболее ранее событие. В зависимости от этого события, может быть сгенерировано новое сообщение, завершена обработка текущего сообщения или сообщение начинает обрабатываться.

Результаты


Widget
Uniform distribution
a: 1
b: 10
Poisson distribution
lambda: 5
Number of messages: 1000
Repeat percentage: 0
Iteration step: 0.1
Generate
Iteration time
Queue length: 8
Event time
Queue length: 8


Widget
Uniform distribution
a: 1
b: 10
Poisson distribution
lambda: 5
Number of messages: 1000
Repeat percentage: 15
Iteration step: 0.02
Generate
Iteration time
Queue length: 49
Event time
Queue length: 64


Widget
Uniform distribution
a: 1
b: 10
Poisson distribution
lambda: 5
Number of messages: 1000
Repeat percentage: 90
Iteration step: 0.02
Generate
Iteration time
Queue length: 792
Event time
Queue length: 774

Листинг

Файл generator.h

```
class Generator
{
public:
    Generator(const int numMsg, const int numRepeat);

    void setEvenDistribution(float _a, float _b);
    double even();

    void setPoissonDistribution(float _lambda);
    int poisson();

    int percentage();
    int eventTime();
    int iterTime(double step);

private:
    int numMessages, perRepeat;
    std::random_device rd;
    std::default_random_engine generator;
    float a, b;
    float lambda;
};
```

Файл generator.cpp

```
Generator::Generator(const int numMsg, const int numRepeat) :
    numMessages(numMsg), perRepeat(numRepeat),
    generator(rd()),
    a(0), b(0),
    lambda(0)
{

}

void Generator::setEvenDistribution(float _a, float _b)
{
    a = _a;
    b = _b;
}

void Generator::setPoissonDistribution(float _lambda)
{
    lambda = _lambda;
}
```

```
double Generator::even()
{
    uniform_real_distribution<double> dist(a, b);
    double num = dist(generator);
    return num;
}

int Generator::poisson()
{
    poisson_distribution<int> dist(lambda);
    int num = dist(generator);
    return num;
}

int Generator::percentage()
{
    mt19937 rng(rd());
    uniform_int_distribution<int> uni(1, 100);
    int num = uni(rng);
    return num;
}
```

```cpp
int Generator::eventTime()
{
    int processedMsg = 0;
    int curLen = 0, maxLen = 0;

    double timeGeneration = even();
    double timeProcessing = timeGeneration + poisson();

    while (processedMsg < numMessages) {
        if (timeGeneration <= timeProcessing) {
            curLen++;
            maxLen = max(maxLen, curLen);
            timeGeneration += even();
        }
        else if (timeGeneration >= timeProcessing) {
            if (curLen > 0) {
                curLen--;
                processedMsg++;
                if (percentage() < perRepeat) {
                    curLen++;
                }
            }

            if (curLen > 0) {
                timeProcessing += poisson();
            }
            else {
                timeProcessing = timeGeneration + poisson();
            }
        }
    }

    return maxLen;
}
```

```cpp
int Generator::iterTime(double step)
{
    int processedMsg = 0;
    int curLen = 0, maxLen = 0;

    double curTime = step;
    double timeGeneration = even();
    double prevTimeGeneration = timeGeneration;
    double timeProcessing = timeGeneration + poisson();

    while (processedMsg < numMessages) {
        if (curTime > timeGeneration) {
            curLen++;
            maxLen = max(maxLen, curLen);
            prevTimeGeneration = timeGeneration;
            timeGeneration += poisson();
        }

        if (curTime > timeProcessing) {
            if (curLen > 0) {
                curLen--;
                processedMsg++;
                if (percentage() <= perRepeat) {
                    curLen++;
                }
            }

            if (curLen > 0) {
                timeProcessing += poisson();
            }
            else {
                timeProcessing = prevTimeGeneration + poisson();
            }
        }
        curTime += step;
    }

    return maxLen;
}
```