Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт Информационных технологий, математики и механики
Кафедра: программной инженерии

Отчет по учебной практике:
Тема:
«Циклический алгоритм управления конфликтными потоками с адаптивной длинной цикла»

Выполнил: студент группы 382003-4м
Кумин Алексей Александрович
Подпись 

Научный руководитель:
Преподаватель
Евгений Владимирович Кудрявцев
Профессор
Михаил Андреевич Федоткин
Подпись

Нижний Новгород
2021 г

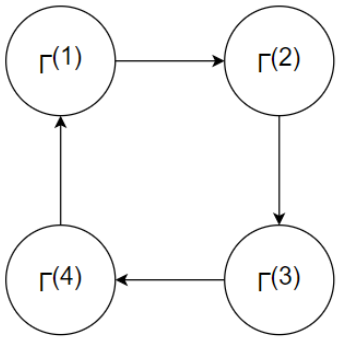
Содержание

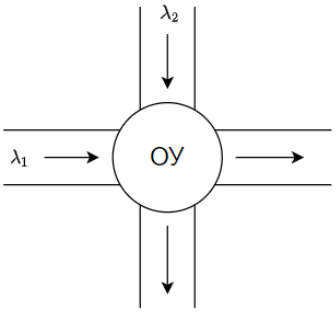
1 Постановка задачи

Необходимо реализовать модель перекрестка с двумя очередями.

Рассмотрим перекресток как обслуживающее устройство, которое имеет 4 состояния, и на обслуживание поступают 2 независимых пуассоновских потока.

Название	Описание	Длительность
$\Gamma^{(1)}$	обслуживание заявок(автомобилей) по первому потоку	случайная величина на отрезке $[T_1, 2T_1]$
$\Gamma^{(2)}$	дообслуживание заявок по первому потоку (обслуживается 1 заявка)	T_2
$\Gamma^{(3)}$	обслуживание заявок(автомобилей) по второму потоку	случайная величина на отрезке $[T_3, 2T_3]$
$\Gamma^{(4)}$	дообслуживание заявок по второму потоку	T_4





Для состояний $\Gamma^{(1)}$, $\Gamma^{(3)}$ длительность выбирается адаптивно, т.е.:

Условие	Изменение времени
1) $k < n_1$	T_1 – не изменяется
2) $n_1 < k < 2 * n_1$	$T' = \frac{T_1}{n_1} k$ – новое время ($T' \in (T_1, 2T_1)$)
3) $2n_1 < k$	$T' = 2 * T_1$ – новое время

где,

n_1 – максимальное количество обслуженных заявок при времени T_1 (состояние $\Gamma^{(1)}$)

k – кол-во заявок в очереди

Аналогично для состояния $\Gamma^{(3)}$.

2 Описание случайных величин

Γ_n	состояние обслуживающего устройства на n-ом шаге
$\varkappa_{1,n}$	длина очереди на n промежутке времени
$\xi_{1,n}$	кол-во заявок, обслуженных в течение n-го промежутка
$\eta_{1,n}$	кол-во заявок, пришедших на n промежутке времени

Уравнение баланса для очереди:

$$\varkappa_{1,n+1} = \varkappa_{1,n} + \eta_{1,n} - \xi_{1,n}. \quad (1)$$

Свойство входного потока:

$$\mathbf{P}\left(\eta_{1,n} = k\right) = \frac{(\lambda t)^k}{k!} e^{-\lambda t} = \varphi_1(k,t). \quad (2)$$

Аналогично и для второго потока.

Теорема 1.

Для случайных величин и элементов $\varkappa_{1,n}$, $\varkappa_{2,n}$ и Γ_n верны следующие рекуррентные соотношения:

$$\Gamma_{n+1} = u(\Gamma_n) = \Gamma^{(n)mod4+1}, \quad (3)$$

$$\varkappa_{1,n+1} = \varkappa_{1,n} + \eta_{1,n} - \xi_{1,n}, \quad (4)$$

$$\varkappa_{2,n+1} = \varkappa_{2,n} + \eta_{2,n} - \xi_{2,n}. \quad (5)$$

Док-во. Первое свойство следует из определения модели, состояния переходят друг из друга - $\Gamma^{(1)} \to \Gamma^{(2)} \to \Gamma^{(3)} \to \Gamma^{(4)} \to \Gamma^{(1)} \to \cdots$. Остальные соотношения следуют из физического определения очереди.

3 Рекуррентные соотношения

3.1 Для потоков по отдельности

Обозначим вероятность: $Q_{r,n}^{(1)}(k) = P\left(\Gamma_n = \Gamma^{(r)}, \varkappa_{1,n} = k\right)$ – вероятность того, что в состоянии $\Gamma^{(r)}$ в первой очереди будет k требований. Аналогично и для второй очереди.

Лемма 1. Вероятности $Q_{r,n}^{(1)}(k)$ будут подчиняться следующим реккурентным соотношениям:

$$Q_{2,n+1}^{(1)}(0) = \sum_{p=0}^{n_1-1} \varphi_1(0, T_1)\, Q_{1,n}^{(1)}(p) + \sum_{p=n_1}^{2n_1} \varphi_1\left(0, \frac{T_1}{n_1} p\right) Q_{1,n}^{(1)}(p) + \\ + \sum_{p=0}^{n_1-1} \sum_{l=1}^{n_1-p} \varphi_1(l, T_1)\, Q_{1,n}^{(1)}(p), \tag{6}$$

$$Q_{2,n+1}^{(1)}(k) = \sum_{p=0}^{n_1} \varphi_1(n_1 + k - p, T_1)\, Q_{1,n}^{(1)}(p) \\ + \sum_{p=n_1}^{2n_1} \varphi_1\left(0, \frac{T_1}{n_1} p\right) Q_{1,n}^{(1)}(p) + \\ + \sum_{p=2n_1+1}^{2n_1+k-1} \varphi_1(2n_1 + k - p, 2T_1)\, Q_{1,n}^{(1)}(p), \tag{7}$$

$$Q_{3,n+1}^{(1)}(k) = \varphi_1(k - p, T_2) Q_{2,n}^{(1)}(0) + \sum_{p=1}^{k+1} \varphi_1(k - p + 1, T_2) Q_{2,n}^{(1)}(p), \tag{8}$$

$$Q_{4,n+1}^{(1)}(k) = \sum_{p=0}^{k} \varphi_1(k - p, T_3') Q_{3,n}^{(1)}(p) \tag{9}$$

$$Q_{1,n+1}^{(1)}(k) = \sum_{p=0}^{k} \varphi_1(k - p, T_4)\, Q_{4,n}^{(1)}(p). \tag{10}$$

Аналогично и для второго потока, только с условием того, что заявки приходят в систему в состоянии 3.

Док-во:

1) Рассмотрим рекуррентные соотношения вероятностей $Q_{r,n}^{(1)}(k)$. В силу независимости случайных величин:и теоремы 1:

$$\begin{aligned}&\mathbf{P}(\varkappa_{n+1}=k)=\mathbf{P}(\varkappa_n=p,\eta_n=l,\xi_n=p+l-k)=\\&=\mathbf{P}(\varkappa_n=p,\eta_n=l)\mathbf{P}\big(\xi_n=p+l-k|(\varkappa_n=p,\eta_n=l)\big)=\\&=\mathbf{P}(\varkappa_n=p)\mathbf{P}(\,\eta_n=l)\mathbf{P}\big(\xi_n=p+l-k|(\varkappa_n=p,\eta_n=l)\big).\end{aligned}\qquad(11)$$

Пусть $\xi_n = 1..N$ – сколько может обслужиться заявок, тогда:

$$\mathbf{P}\big(\xi_n=N|(\varkappa_n=p,\eta_n=l)\big)=1,\qquad p<N,k+l>N,$$

$$\mathbf{P}\big(\xi_n=k+l|(\varkappa_n=p,\eta_n=l)\big)=1,\qquad p<N,k+l<N,$$

$$\mathbf{P}\big(\xi_n=N|(\varkappa_n=p,\eta_n=l)\big)=1,\qquad p>N,$$

$$\mathbf{P}\big(\xi_n=m\ \neq N|(\varkappa_n=p,\eta_n=l)\big)=0,\qquad p>N.$$

2) Рассмотрим при каких условиях в состоянии $\Gamma^{(1)}$ будет изменяться время обслуживания T_1:

   В силу теоремы 1: $\varkappa_{1,n+1}=\varkappa_{1,n}+\eta_{1,n}-\xi_{1,n}$,

   n_1 – максимальное количество обслуженных заявок при времени T_1

1) $k < n_1$	T_1 – не изменяется	$\varkappa_{1,n+1}=\varkappa_{1,n}+\eta_{1,n}-\xi_{1,n}=$ $=\max(0,\varkappa_{1,n}+\eta_{1,n}-n_1)$
2) $n_1 < k < 2*n_1$	$T'=\frac{T_1}{n_1}\varkappa_{1,n}$ – новое время	$\varkappa_{1,n+1}=\varkappa_{1,n}+\eta_{1,n}-\xi_{1,n}=$ $=\varkappa_{1,n}+\eta_{1,n}-\frac{\varkappa_{1,n}*n_1}{n_1}=$ $=\eta_{1,n}$
3) $2n_1 < k$	$T'=2*T_1$	$\varkappa_{1,n+1}=\varkappa_{1,n}+\eta_{1,n}-\xi_{1,n}=$ $=\varkappa_{1,n}+\eta_{1,n}-2*n_1$

3) Теперь рассмотрим, как будут вести себя $\varkappa_{1,n},\eta_{1,n},\xi_{1,n}$ при различных k:

$k=0$	$\varkappa_{1,n}=0,\eta_{1,n}=0,\xi_{1,n}=0.$
	$\varkappa_{1,n}=p,\eta_{1,n}=0,\xi_{1,n}=p\ (p\le 2n_1).$
	$\varkappa_{1,n}=p,\eta_{1,n}=l,\xi_{1,n}=p+l\ (p+l\le n_1).$
$k>0$	$\varkappa_{1,n}=p,\eta_{1,n}=l,\xi_{1,n}=p+l-k=n_1\ (p\le n_1,l=n_1+k-p).$
	$\varkappa_{1,n}=p,\eta_{1,n}=k,\xi_{1,n}=p\ (n_1<p\le 2n_1).$
	$\varkappa_{1,n}=p,\eta_{1,n}=l,\xi_{1,n}=p+l-k=$ $=2n_1\ (l=2n_1+k\ -p\ge 0,p\le 2n_1+k).$

4) Из данных выкладок получаем рекуррентные соотношения в состояниях, суммируя вероятности предыдущих шагов

   Аналогично составляются соотношения для второго потока.

3.2 Для потоков в совокупности

Обозначим вероятность: $Q_{r,n}(k,l) = \mathbf{P}\big(\Gamma_n = \Gamma^{(r)}, \varkappa_{1,n} = k, \varkappa_{2,n} = l\big)$.

Теорема 2.

Вероятности $Q_{r,n}(k,l)$ удовлетворяют следующим рекуррентным соотношениям:

$$\begin{aligned} Q_{2,n+1}(0,l) = &\sum_{p=0}^{n_1-1} \varphi_1(0,T_1) \sum_{q=0}^{l} \varphi_2(l-q,T_1) Q_{1,n}(p,q) + \\ &+ \sum_{p=n_1}^{2n_1} \varphi_1\left(0,\frac{T_1}{n_1}p\right) \sum_{q=0}^{l} \varphi_2\left(l-q,\frac{T_1}{n_1}p\right) Q_{1,n}(p,q) + \\ &+ \sum_{p=0}^{n_1-1} \sum_{m=1}^{n_1-p} \varphi_1(m,T_1) \sum_{q=0}^{l} \varphi_2(l-q,T_1)\, Q_{1,n}(p,q), \end{aligned} \tag{12}$$

$$\begin{aligned} Q_{2,n+1}(k,l) = &\sum_{p=0}^{n_1} \varphi_1(n_1+k-p,T_1) \sum_{q=0}^{l} \varphi_2(l-q,T_1) Q_{1,n}(p,q) + \\ &+ \sum_{p=n_1}^{2n_1} \varphi_1\left(k,\frac{T_1}{n_1}p\right) \sum_{q=0}^{l} \varphi_2\left(l-q,\frac{T_1}{n_1}p\right) Q_{1,n}(p,q) + \\ &+ \sum_{p=2n_1+1}^{2n_1+k-1} \varphi_1(2n_1+k-p,2T_1) \sum_{q=0}^{l} \varphi_2(l-q,2T_1)\, Q_{1,n}(p,q), \end{aligned} \tag{13}$$

$$\begin{aligned} Q_{3,n+1}(k,l) = &\varphi_1(k,T_2) \sum_{q=0}^{l} \varphi_2(l-q,T_2) Q_{2,n}(0,q) + \\ &+ \sum_{p=1}^{k+1} \varphi_1(k-p+1,T_2) \sum_{q=0}^{l} \varphi_2(l-q,T_2)\, Q_{2,n}(p,q), \end{aligned} \tag{14}$$

$$\begin{aligned} Q_{4,n+1}(k,0) = &\sum_{q=0}^{n_2-1} \varphi_2(0,T_3) \sum_{p=0}^{k} \varphi_2(k-p,T_3) Q_{3,n}(p,q) + \\ &+ \sum_{q=n_2}^{2n_2} \varphi_2\left(0,\frac{T_3}{n_2}q\right) \sum_{p=0}^{k} \varphi_1\left(k-p,\frac{T_3}{n_2}q\right) Q_{3,n}(p,q) + \\ &+ \sum_{q=0}^{n_2-1} \sum_{m=1}^{n_2-q} \varphi_2(m,T_3) \sum_{p=0}^{k} \varphi_1(k-p,T_3)\, Q_{3,n}(p,q), \end{aligned} \tag{15}$$

$$Q_{4,n+1}(k,l) = \sum_{q=0}^{n_2} \varphi_2(n_2 + l - q, T_3) \sum_{p=0}^{k} \varphi_1(k-p, T_3) Q_{3,n}(p,q) +$$
$$+ \sum_{q=n_1}^{2n_2} \varphi_2\left(l, \frac{T_3}{n_2} p\right) \sum_{p=0}^{k} \varphi_1\left(k-p, \frac{T_3}{n_2} q\right) Q_{3,n}(p,q) + \quad (16)$$
$$+ \sum_{q=2n_2+1}^{2n_2+l-1} \varphi_2(2n_2 + l - q, 2T_3) \sum_{p=0}^{k} \varphi_1(k-p, 2T_3)\, Q_{3,n}(p,q),$$

$$Q_{1,n+1}(k,l) = \varphi_2(k, T_4) \sum_{p=0}^{k} \varphi_1(k-p, T_4) Q_{4,n}(p,0) + \quad (17)$$
$$+ \sum_{q=1}^{l+1} \varphi_2(l - q + 1, T_4) \sum_{p=0}^{k} \varphi_1(k-p, T_4)\, Q_{4,n}(p,q).$$

Док-во: рекуррентные соотношения следуют из леммы 1.

3.3 Марковость системы

Теорема 3. .Случайная векторная последовательность $(\Gamma_n, \varkappa_{1,n}, \varkappa_{2,n})$ с начальным состоянием $(\Gamma_0, \varkappa_{1,0}, \varkappa_{2,0})$ является марковской.

Док-во. Для доказательства нам потребуется показать, что последовательность удовлетворяет марковскому свойству, а именно:

$$\begin{gathered}\mathbf{P}((\Gamma_n = \Gamma^{(r_n)}, \varkappa_{1,n} = k_n, \varkappa_{2,n} = l_n)| \\ |(\Gamma_{n-1} = \Gamma^{(r_{n-1})}, \varkappa_{1,n-1} = k_{n-1}, \varkappa_{2,n-1} = l_{n-1}), \dots \\ \dots, (\Gamma_0 = \Gamma^{(r_0)}, \varkappa_{1,0} = k_1, \varkappa_{2,0} = l_0)) = \\ = \mathbf{P}((\Gamma_n = \Gamma^{(r_n)}, \varkappa_{1,n} = k_n, \varkappa_{2,n} = l_n)| \\ |(\Gamma_{n-1} = \Gamma^{(r_{n-1})}, \varkappa_{1,n-1} = k_{n-1}, \varkappa_{2,n-1} = l_{n-1}))\,.\end{gathered} \tag{18}$$

По формуле полной вероятности имеем:

$$\begin{gathered}\mathbf{P}((\Gamma_n = \Gamma^{(r_n)}, \varkappa_{1,n} = k_n, \varkappa_{2,n} = l_n)| \\ |(\Gamma_{n-1} = \Gamma^{(r_{n-1})}, \varkappa_{1,n-1} = k_{n-1}, \varkappa_{2,n-1} = l_{n-1}), \dots \\ \dots, (\Gamma_0 = \Gamma^{(r_0)}, \varkappa_{1,0} = k_1, \varkappa_{2,0} = l_0)) = \\ \sum_{m_1=0}^{\infty}\sum_{m_2=0}^{\infty} \mathbf{P}\,((u(\Gamma^{(r_{n-1})}) = \Gamma^{(r_n)}, \\ , v_1\big(\Gamma^{(r_{n-1})}, k_{n-1}, m_1\big) = k_n, \\ , v_2(\Gamma^{(r_{n-1})}, l_{n-1}, m_2) = l_n)| \\ |(\Gamma_{n-1} = \Gamma^{(r_{n-1})}, \varkappa_{1,n-1} = k_{n-1}, \varkappa_{2,n-1} = l_{n-1}, \\ , \eta_{1,n-1} = m_1, \eta_{2,n-1} = m_2), \dots \\ \dots, \big(\Gamma_0 = \Gamma^{(r_0)}, \varkappa_{1,0} = k_1, \varkappa_{2,0} = l_0\big)).\end{gathered} \tag{19}$$

Видно, что условие $(u\big(\Gamma^{(r_{n-1})}\big) = \Gamma^{(r_n)}, \mathrm{v}_1\big(\Gamma^{(r_{n-1})}, k_{n-1}, m_1\big) = k_n, \mathrm{v}_2\big(\Gamma^{(r_{n-1})}, l_{n-1}, m_2\big) = l_n)$ не зависит от $\big(\Gamma_0 = \Gamma^{(r_0)}, \varkappa_{1,0} = k_0, \varkappa_{2,0} = l_0\big) \dots \big(\Gamma_{n-2} = \Gamma^{(r_{n-2})}, \varkappa_{1,\mathrm{n}-2} = k_{n-2}, \varkappa_{2,n-2} = l_{n-2}\big)$, поэтому данная вероятность равна:

$$\begin{gathered}\sum_{m_1=0}^{\infty}\sum_{m_2=0}^{\infty} \mathbf{P}\,(u\big(\Gamma^{(r_{n-1})}\big) = \Gamma^{(r_n)}, \\ , v_1\big(\Gamma^{(r_{n-1})}, k_{n-1}, m_1\big) = k_n, \\ , v_2\big(\Gamma^{(r_{n-1})}, l_{n-1}, m_2\big) = l_n)| \\ |(\Gamma_{n-1} = \Gamma^{(r_{n-1})}, \varkappa_{1,n-1} = k_{n-1}, \\ , \varkappa_{2,n-1} = l_{n-1}, \eta_{1,n-1} = m_1, \eta_{2,n-1} = m_2)).\end{gathered} \tag{20}$$

Аналогично выводится, что

$$\begin{gathered}\mathbf{P}((\Gamma_n = \Gamma^{(r_n)}, \varkappa_{1,n} = k_n, \varkappa_{2,n} = l_n)| \\ |(\Gamma_{n-1} = \Gamma^{(r_{n-1})}, \varkappa_{1,n-1} = k_{n-1}, \varkappa_{2,n-1} = l_{n-1}), \dots \\ \dots, (\Gamma_0 = \Gamma^{(r_0)}, \varkappa_{1,0} = k_1, \varkappa_{2,0} = l_0)) = \\ \sum_{m_1=0}^{\infty} \sum_{m_2=0}^{\infty} \mathbf{P}\,(u\left(\Gamma^{(r_{n-1})}\right) = \Gamma^{(r_n)}, \\ , v_1\left(\Gamma^{(r_{n-1})}, k_{n-1}, m_1\right) = k_n, \\ , v_2\left(\Gamma^{(r_{n-1})}, l_{n-1}, m_2\right) = l_n)| \\ |(\Gamma_{n-1} = \Gamma^{(r_{n-1})}, \varkappa_{1,n-1} = k_{n-1}, \\ , \varkappa_{2,n-1} = l_{n-1}, \eta_{1,n-1} = m_1, \eta_{2,n-1} = m_2)).\end{gathered} \tag{21}$$

Таким образом получаем, что левые и правые части равенства (18) совпадают. Таким образом, теорема доказана.

4 Имитационное моделирование

Исследуем задачу с помощью программы, имитирующей перекресток и реализующей циклический алгоритм, а так же алгоритм с адаптивной длинной цикла. Описание программы:

Исходные данные	T1, T2, T3, T4	время нахождения системы в каждом из состояний
	l1, l2	интенсивность потоков
	t1, t2	время обслуживания одной заявки по потокам
	x1, x2	начальное кол-во заявок в очередях
	N	количество шагов системы
Выходные данные	midX1, midX2	среднее число заявок в очередях за время действия системы
	mid T x1, mid T x2	среднее время ожидания заявки в очередях
	таблица состояний в каждый момент времени (T1, T2, T3, T4)	
	количество заявок в очередях в эти моменты времени.	
Контроль T1	Адаптивный выбор длительности состояния $\Gamma^{(1)}$ (из промежутка [T1, 2*T1])	
Контроль T3	Адаптивный выбор длительности состояния $\Gamma^{(3)}$ (из промежутка [T3, 2*T3])	

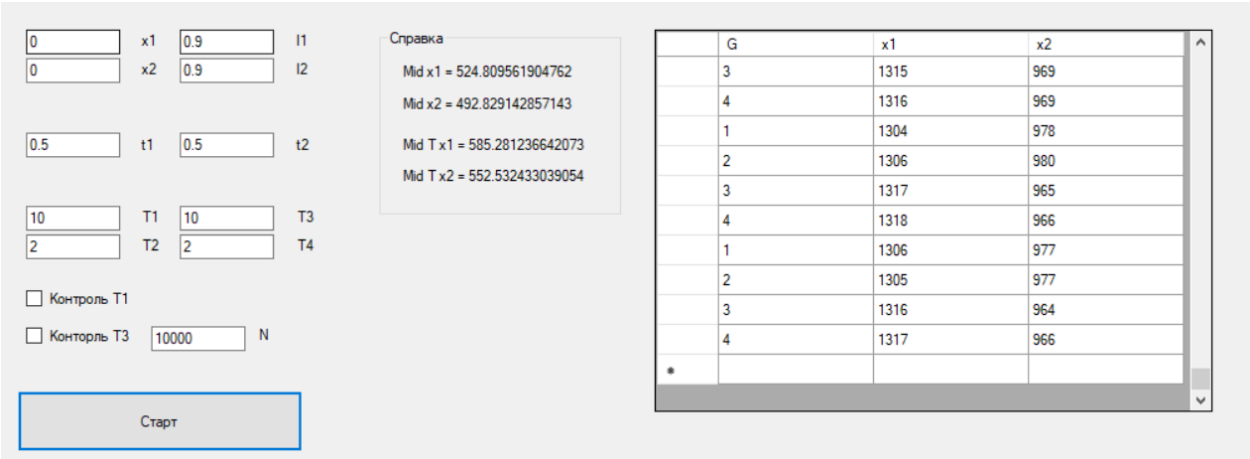


Рис.1

На Рис.1 видно, что при выбранных параметрах очереди неограниченно растут. Среднее время ожидания заявок и среднее кол-во заявок возрастают с течением времени.

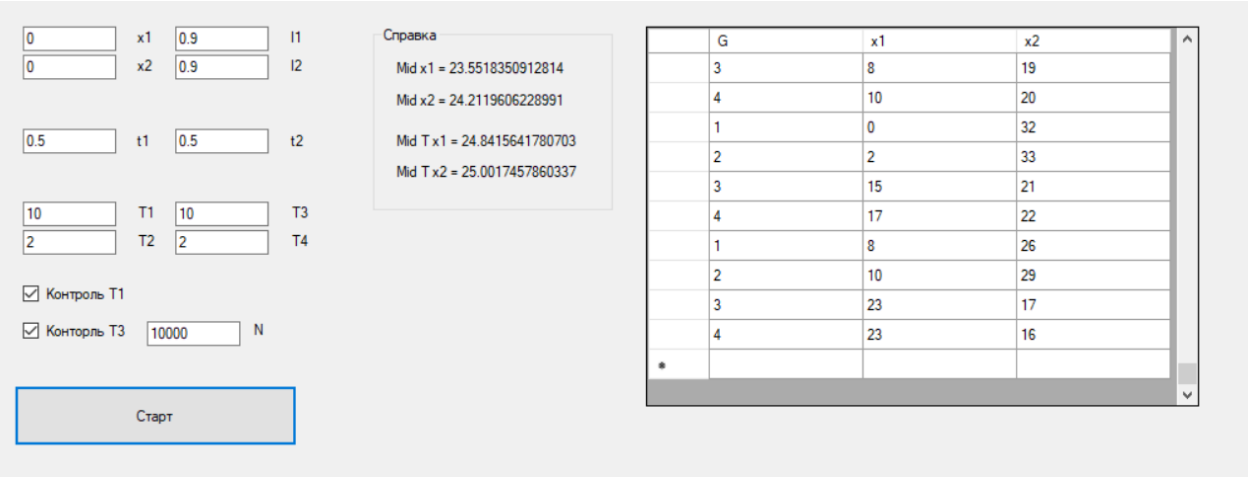


Рис.2

На Рис. 2 видна демонстрация работы циклического алгоритма с адаптивной длинной цикла – за это отвечает контроь времени T1 и T3 – в системе наблюдается стационар.

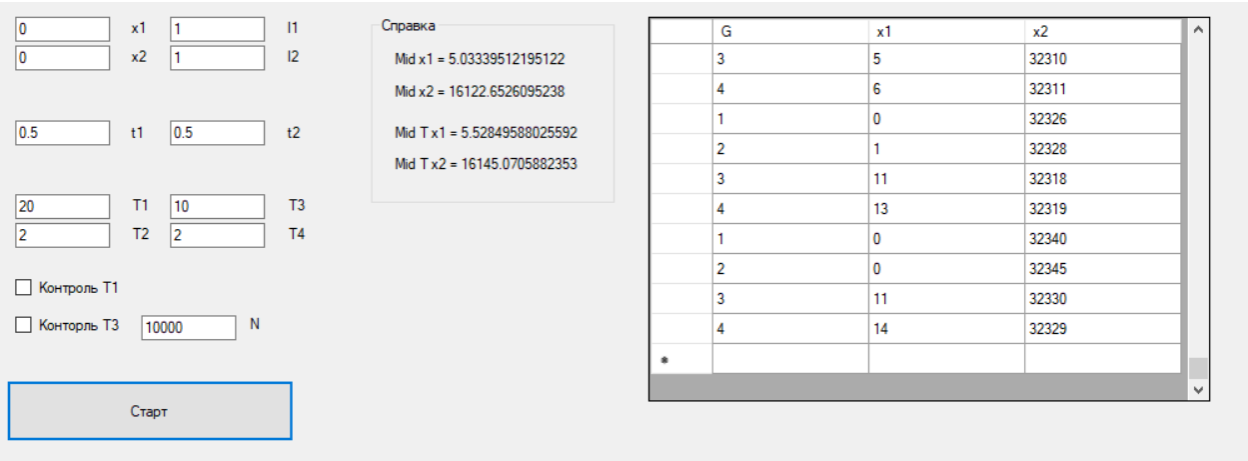


Рис.3

На Рис. 3 наблюдается по одному потоку в связи с увеличением времени T1 – времени нахождения в первом состоянии системы.

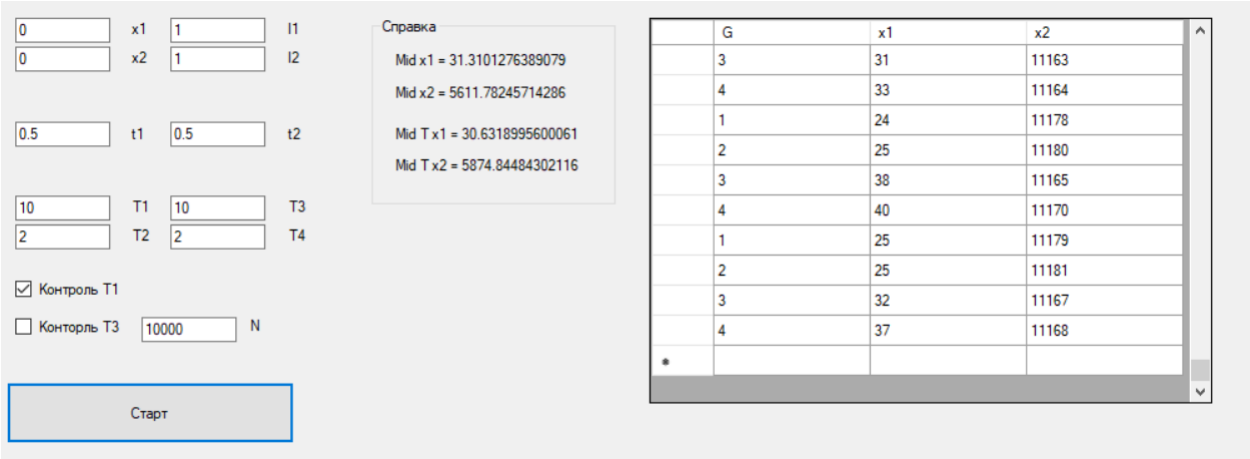


Рис.4

На Рис.4 Так же наблюдается стационар по первому потоку – адаптицая T1

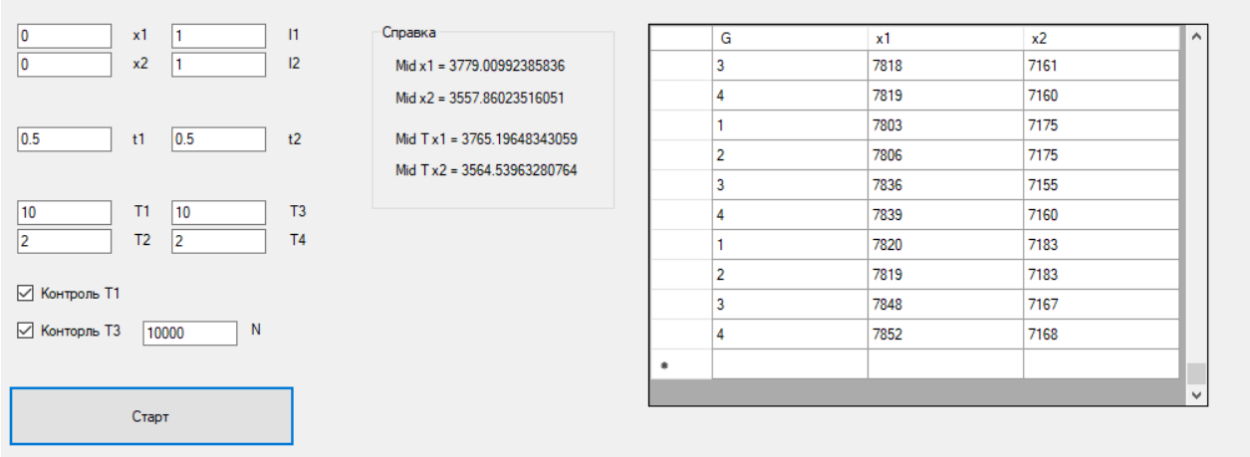


Рис.5

Рис.5 – видно, что при некоторых наборах параметров адаптация не может привести систему в стационар

Вывод: можно подобрать такой набор параметров, при котором очереди не будут увеличиваться и в системе будет наблюдаться стационар, т.е. размер очередей будет постоянным. Также, стационар можно наблюдать в некоторых случаях при адаптировании времени T1, T3.

5 Алгоритмы

5.1 Моделирование случайных величин

5.1.1 1 способ

1) Моделируем методом обратной функции случайную величину, имеющую показательное распределение с параметром a – время от появления в очереди одной заявки до появления следующей
2) Пока сумма этих случайных величин меньше заданного промежутка времени, проводим подсчет кол-ва генераций новых случайных величин и прибавляем их к сумме
3) Выводим кол-во генераций случайных величин

```
double relX(double a)
{
	return (-log(1 - ((double)(rand() + 0.5) / (double)RAND_MAX)) / a);
}

int relETA(double a, double t)
{
	double sum = relX(a);
	int i = 0;
	while (t > sum)
	{
		sum += relX(a);
		i++;
	}
	return i;
}
```

5.1.2 2 способ

1) Вычислим значение exp(a*t)*(вероятность, что придет к заявок)
2) Пока эта величина больше суммы первых множителей пуассоновской случайной величины, продолжаем суммировать и запоминаем кол-во членов ряда
3) Выводим кол-во членов этого ряда -1

```
long int fact(int N)
{
	if (N == 1 || N == 0)
		return 1;
	return N * fact(N - 1);
}

int relETA1(double a, double t)
{
	double eta = exp(a * t) *(double)rand() / (double)RAND_MAX;
	double sum = 0;
	int i = 0;
	while (eta > sum)
	{
		sum += pow(a * t, i) / (double)fact(i);
		i++;
	}
	return i - 1;
}
```

5.1.3 Общее моделирование

При больших значениях a*t 2 способ будет работать долго и некорректно из-за того ему приходится возводить это число в степень и вычислять факториалы и экспоненты, однако, этот способ хорош для вычисления пуассоновской случайной величины на при малых значениях a и t, поэтому выделим 2 случая

```
int Eta(double a, double t)
{
	if ((a * t) < 20)
		return relETA1(a, t);
	else
		return relETA(a, t);
}
```

5.2 Класс для моделирования потока StreamModel

5.2.1 StreamModel.h

```
#pragma once
#include <vector>
#include <iostream>
#include <random>
#include <time.h>
#include <math.h>
using namespace std;

class StreamModel
{
public:
	int key;//состояние
	int x;//сколько было в очереди
	double t; //время обработки заявки
	double l;//интенстивность потока
	vector<int> Auto;
	vector<double> T;
	vector<double> mesX;//массив кол-ва заявок в разные промежутки времени

	StreamModel(int KEY = 0, int X = 0, double T = 1, double L = 1);

	void move(double T);

	void service(double T);

	void stagnation(double T);

	double Med_x();

	double Med_Tx();

	~StreamModel();
};
```

5.2.2 Простой потока

```
void StreamModel::stagnation(double T1)
{
	int ETA = Eta(l, T1);
	for (int i = 0; i < T.size(); i++)
		if (Auto[i] == 1)
			T[i] += T1;
	for (int i = 0; i < (ETA); i++)
	{
		Auto.push_back(1);
		T.push_back(0);
	}
	//....
	x += ETA;
}
```

5.2.3 Дообслуживание одной заявки

```
void StreamModel::service(double T1)
{
	if (T1 >= t)
	{
		int ETA = Eta(l, T1);
		if ((x + ETA - 1) >= 0)
		{
			//.....
			int k = 1;
			for (int i = 0; i < Auto.size(); i++)
			{
```

```
                        if (Auto[i] == 1)
                        {
                                Auto[i] = 0;
                                k = 0;
                                break;
                        }
                }
                for (int i = 0; i < T.size(); i++)
                {
                        if (Auto[i] == 1)
                                T[i] += T1;
                }
                for (int i = 0; i < (ETA); i++)
                {
                        Auto.push_back(1);
                        T.push_back(0);
                }
                if (k)
                {
                        for (int i = 0; i < Auto.size(); i++)
                        {
                                if (Auto[i] == 1)
                                {
                                        Auto[i] = 0;
                                        break;
                                }
                        }
                }
                //.....

                x += ETA - 1;
        }
        else
        {
                //......
                for (int i = 0; i < T.size(); i++)
                        if (Auto[i] == 1)
                                T[i] += T1;
                for (int i = 0; i < (ETA); i++)
                {
                        Auto.push_back(1);
                        T.push_back(0);
                }
                //......

                x += ETA;
        }
    }
    mesX.push_back(x);
}
```

5.2.4 Обслуживание потока

```
void StreamModel::move(double T1)
{
    double sumT = 0;
    while (sumT < T1)
    {
        sumT += t;
        service(t);
    }
}
```

5.2.5 Вычисление средний длинны очереди и среднего ожидания обслуживания

```
double Med(vector<double> mes)
{
	double Sum = 0;
	for (int i = 0; i < mes.size(); i++)
		Sum += mes[i];
	return (Sum / (double)mes.size());
}

double StreamModel::Med_x()
{
	return Med(mesX);
}

double StreamModel::Med_Tx()
{
	return Med(T);
}
```

6 Литература

1) Зорин А.В, Зорин В.А, Федоткин М.А. «Теория управляемых систем массового обслуживания: Учебное пособие.» Нижний Новгород: Издательство Нижегородского госуниверситета, 2007 г. – 47 с.
2) Гнеденко Б.В., Коваленко И.Н. «Введение в теорию массового обслуживания» М.: Наука, 1966. — 432 с.
3) Зорин А.В, Зорин В.А, Федоткин М.А «Моделирование случайных величин и проверка гипотез о виде распределения» Учебно-методическое пособие. — Нижний Новгород: Нижегородский госуниверситет, 2017. — 19 с.
4) Некруткин В.В «Моделирование распределений» Материалы специального курса и специального семинара 4 февраля 2013 г. – 90 с

7 Приложение

7.1 Файл StreamModel.h

```
#pragma once
#include <vector>
#include <iostream>
#include <random>
#include <time.h>
#include <math.h>
using namespace std;
class StreamModel
{
public:
int key;//состояние
int x;//сколько было в очереди
double t; //время обработки заявки
double l;//интенстивность потока
vector<int> Auto;
vector<double> T;
vector<double> mesX;//массив кол-ва заявок в разные промежутки времени
StreamModel(int KEY = 0, int X = 0, double T = 1, double L = 1);
void move(double T);
void service(double T);
void stagnation(double T);
double Med_x();
double Med_Tx();
~StreamModel();
};
```

7.2 Файл StreamModel.cpp

```
#include "stdafx.h"
#include "StreamModel.h"
using namespace std;
long int fact(int N)
{
if (N == 1 || N == 0)
return 1;
return N * fact(N - 1);
}
double relX(double a)
{
return (-log(1 - ((double)(rand() + 0.5) / (double)RAND_MAX)) / a);
}
int relETA(double a, double t)
{
double sum = relX(a);
int i = 0;
while (t > sum)
{
sum += relX(a);
i++;
```

```
}
return i;
}
int relETA1(double a, double t)
{
double eta = exp(a * t) *(double)rand() / (double)RAND_MAX;
double sum = 0;
int i = 0;
while (eta > sum)
{
sum += pow(a * t, i) / (double)fact(i);
i++;
}
16
return i - 1;
}
int Eta(double a, double t)
{
if ((a * t) < 20)
return relETA1(a, t);
else
return relETA(a, t);
}
double Med(vector<double> mes)
{
double Sum = 0;
for (int i = 0; i < mes.size(); i++)
Sum += mes[i];
return (Sum / (double)mes.size());
}
//Реализация.....................................................
StreamModel::StreamModel(int KEY, int X, double T, double L)
{
key = KEY;
x = X;
t = T;
l = L;
}
void StreamModel::move(double T1)
{
double sumT = 0;
while (sumT < T1)
{
sumT += t;
service(t);
}
}
void StreamModel::service(double T1)
{
if (T1 >= t)
{
int ETA = Eta(l, T1);
if ((x + ETA - 1) >= 0)
```

```
{
//.....
int k = 1;
for (int i = 0; i < Auto.size(); i++)
{
if (Auto[i] == 1)
{
Auto[i] = 0;
k = 0;
break;
}
}
for (int i = 0; i < T.size(); i++)
{
if (Auto[i] == 1)
T[i] += T1;
}
for (int i = 0; i < (ETA); i++)
{
Auto.push_back(1);
T.push_back(0);
}
if (k)
{
for (int i = 0; i < Auto.size(); i++)
{
if (Auto[i] == 1)
{
Auto[i] = 0;
break;
}
}
}
//.....
17
x += ETA - 1;
}
else
{
//......
for (int i = 0; i < T.size(); i++)
if (Auto[i] == 1)
T[i] += T1;
for (int i = 0; i < (ETA); i++)
{
Auto.push_back(1);
T.push_back(0);
}
//......
x += ETA;
}
}
mesX.push_back(x);
```

```
}
void StreamModel::stagnation(double T1)
{
int ETA = Eta(l, T1);
for (int i = 0; i < T.size(); i++)
if (Auto[i] == 1)
T[i] += T1;
for (int i = 0; i < (ETA); i++)
{
Auto.push_back(1);
T.push_back(0);
}
//....
x += ETA;
}
double StreamModel::Med_x()
{
return Med(mesX);
}
double StreamModel::Med_Tx()
{
return Med(T);
}
StreamModel::~StreamModel()
{
}
```

7.3 Файл Form1.h(главная часть)

```
void pushTab(System::Windows::Forms::DataGridView^ Tab, int i, int key, int x1, int x2)
{
dataGridView1->Rows->Add();
dataGridView1->Rows[i]->Cells[0]->Value = key;
dataGridView1->Rows[i]->Cells[1]->Value = x1;
dataGridView1->Rows[i]->Cells[2]->Value = x2;
}
double controlT(StreamModel G, double T0, double maxT)
{
double T;
if (G.x > (int)(T0 / G.t + 1))
{
if ((G.x * G.t) < (maxT))
T = (G.x * G.t);
else
T = maxT;
}
else
T = T0;
return T;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
int x1 = Convert::ToInt16(textBox1->Text);
int x2 = Convert::ToInt16(textBox2->Text);
```

```
double L1 = Convert::ToDouble(textBox3->Text);
18
double L2 = Convert::ToDouble(textBox4->Text);
double G1t = Convert::ToDouble(textBox5->Text);
double G2t = Convert::ToDouble(textBox11->Text);
int N = Convert::ToInt16(textBox10->Text);
double maxT = Convert::ToInt16(textBox12->Text);
vector<int> Tab1, Tab2, Tab3;
vector<double> T(4);
T[0] = Convert::ToDouble(textBox6->Text);
T[1] = Convert::ToDouble(textBox7->Text);
T[2] = Convert::ToDouble(textBox9->Text);
T[3] = Convert::ToDouble(textBox8->Text);
vector<double> T1(2);
T1[0] = T[0];
T1[1] = T[2];
StreamModel G1(1, x1, G1t, L1);
StreamModel G2(1, x2, G2t, L2);
for (int p, i = 0; i < N; i++)
{
p = i % 4;
G1.key = p + 1;
G2.key = p + 1;
switch (p)
{
case 0:
{
G1.move(T[p]);
if (checkBox1->Checked)
T[p] = controlT(G1, T1[0], maxT);
G2.stagnation(T[p]);
break;
}
case 1:
{
G1.service(T[p]);
G2.stagnation(T[p]);
break;
}
case 2:
{
G2.move(T[p]);
if (checkBox2->Checked)
T[p] = controlT(G2, T1[1], maxT);
G1.stagnation(T[p]);
break;
}
case 3:
{
G1.stagnation(T[p]);
G2.service(T[p]);
break;
}
```

```
}
Tab1.push_back(G1.key);
Tab2.push_back(G1.x);
Tab3.push_back(G2.x);
}
dataGridView1->Rows->Clear();
for (int i = 0; i < N; i++)
{
pushTab(dataGridView1, i, Tab1[i], Tab2[i], Tab3[i]);
}
label11->Text = "Mid x1 = " + Convert::ToString(G1.Med_x()) + " ";
label12->Text = "Mid x2 = " + Convert::ToString(G2.Med_x()) + " ";
label13->Text = "Mid T x1 = " + Convert::ToString(G1.Med_Tx()) + " ";
label14->Text = "Mid T x2 = " + Convert::ToString(G2.Med_Tx()) + " ";
}
};
```