Белорусский государственный университет информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Отчет по лабораторным работам № 5, 6

по дисциплине «Системный анализ и машинное моделирование»

Вариант 3в

Проверил:

Мельник Н. И.

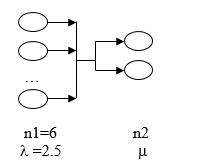
Выполнил:

студент группы 651001

Арабей Д.И.

**1. Построение и исследование аналитической модели непрерывно-стохастической системы массового обслуживания**

СМО с ожиданием ответа



Определить среднее число ожидающих ответа источников, среднее время ожидания ответа и абсолютную пропускную способность (интенсивность на выходе системы) Входные потоки и потоки обслуживаний – простейшие.

в) n2=1, μ=18

Решение в Mathcad:

































**2. Построение и исследование имитационной модели непрерывно-стохастической системы массового обслуживания**

* для заданной СМО простроить имитационную модель и исследовать ее:

Результаты работы программы:



**Вывод**:

В результате лабораторной работы была аналитически смоделирована непрерывно-стохастическая СМО и разработана программа, имитирующая ее поведение. Построенная модель позволяет статистически подсчитать необходимые характеристики СМО такие, как среднее число ожидающих ответа источников, среднее время ожидания ответа и абсолютную пропускную способность.

**Код программы**:

type Generator struct {

    Generated int

    Attempted int

    chance float64

    CurrentItem \*Item

}

func NewGenerator(lambda float64) \*Generator {

    return &Generator{

        Generated: 0,

        chance: lambda / 60,

    }

}

func (g \*Generator) MaybeGenerate() \*Item {

    g.Attempted += 1

    attempt := rand.Float64()

    if g.chance > attempt {

        g.Generated += 1

        g.CurrentItem = &Item{

            ID: g.Generated,

            ProcessingTime: 0,

            Parent: g,

            ProcessingRequired: ProcessingRequired,

            Waiting: 1,

        }

        return g.CurrentItem

    }

    return nil

}

func (g \*Generator) HasItem() bool {

    return g.CurrentItem != nil

}

func (g \*Generator) IsBlocked() bool {

    return g.HasItem()

}

type Handler struct {

    ID int

    CurrentItem \*Item

    Processed int

    Attempted int

    chance float64

}

func (h \*Handler) MaybeHandle() bool {

    h.CurrentItem.ProcessingTime++

    attem := rand.Float64()

    if attem < 0.3 {

        h.Processed++

        h.GetItem().Processed()

        h.SetItem(nil)

        return true

    }

    return false

}

func (h \*Handler) SetItem(item \*Item) \*Handler {

    h.CurrentItem = item

    return h

}

func (h \*Handler) GetItem() \*Item {

    return h.CurrentItem

}

func (h \*Handler) HasItem() bool {

    return h.CurrentItem != nil

}

type Queue struct {

    MaxLength int

    Items []\*Item

}

func (q \*Queue) Length() int {

    return len(q.Items)

}

func (q \*Queue) Push(item \*Item) bool {

    if len(q.Items) < q.MaxLength {

        q.Items = append(q.Items, item)

        return true

    }

    return false

}

func (q \*Queue) Pop() \*Item {

    if len(q.Items) == 0 {

        return nil

    }

    item := q.Items[len(q.Items)-1]

    q.Items = q.Items[:len(q.Items)-1]

    return item

}

func (q \*Queue) IsFull() bool {

    return len(q.Items) == q.MaxLength

}

var (

    ticks = 100000

    generatorsLambda = 2.5

    generatorsCount = 6

    handlersCount = 1

)

func main() {

    rand.Seed(time.Now().UTC().UnixNano())

    generators := make([]\*src.Generator, 0)

    for i := 0; i < generatorsCount; i++ {

        generators = append(generators, src.NewGenerator(generatorsLambda))

    }

    handlers := make([]\*src.Handler, 0)

    for i := 0; i < handlersCount; i++ {

        handlers = append(handlers, &src.Handler{

            ID: i,

            CurrentItem: nil,

        })

    }

    system := &src.System{

        Handlers: handlers,

        Generators: generators,

        ProcessedItems: make([]\*src.Item, 0),

    }