**Комп’ютерний практикум 2**

**Розробка простої моделі масового обслуговування**

**2.1 Завдання до практичної роботи**

1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм (рис 2.1). **10 балів.**

PROCESS 1

DESPOSE

CREATE

Рисунок 2.1 – Схема моделі.

1. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою. **30 балів.**
2. Модифікувати алгоритм, додавши обчислення середнього очікування в черзі. **30 балів.**
3. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці. **30 балів.**

**2.2 Теоретичні відомості з алгоритмів імітації дискретно-подійних систем**

Алгоритм, який відтворює функціонування системи, за допомогою комп’ютерної програми називається *алгоритмом імітації*.

Побудова алгоритму імітації складається з побудови:

1. алгоритму просування модельного часу;
2. алгоритму просування стану моделі в залежності від часу;
3. алгоритму збирання інформації про поведінку моделі у процесі імітації.

Просування модельного часу та просування стану моделі в залежності від часу становлять єдину задачу побудови алгоритму імітації. Якщо дослідник для реалізації алгоритму імітації використовує імітаційну мову моделювання, то моделюючий алгоритм закладений в самий засіб реалізації, і досліднику тільки корисно знати як він здійснюється, щоб не допустити помилки у застосуванні тієї або іншої імітаційної мови. Якщо ж дослідник використовує універсальну мову програмування, то задачу побудови моделюючого алгоритму йому потрібно вирішувати самостійно.

Узагальнена схема алгоритму імітації представлена на рисунку 2.2

Введення початкових даних

Доки

поточний час < час моделювання

Просунути поточний час

Виконати змінювання стану моделі

Виведення результатів моделювання

Рисунок 2.2 – Узагальнена схема алгоритму імітації.

Розглянемо відомі способи побудови алгоритмів імітації.

Способи побудови алгоритму просування модельного часу

Існують три способи просування модельного часу:

* за принципом *Δt*,
* за принципом найближчої події,
* за принципом послідовного проведення об’єктів уздовж моделі.

###### *Принцип Δt*

Весь інтервал часу, протягом якого моделюється система, поділяється на рівні інтервали довжиною *Δt=const*. При кожному просуванні модельного часу на величину *Δt* послідовно визначаються всі зміни, що відбуваються в моделі. Величина одного інтервалу *Δt* повинна бути настільки малою, щоб в одному інтервалі *Δt* відбувалось не більш однієї події. В противному випадку логіка алгоритму імітації порушується і функціонування моделі суттєво відрізняється від функціонування реальної системи.

Принцип *Δt* є найбільш універсальним, але частіше він використовується для моделювання неперервних динамічних систем, оскільки вимагає у порівнянні з іншими способами більших затрат комп’ютерного часу при тій же точності моделювання.

Принцип найближчої події

Дискретні системи, імітаційне моделювання яких розглядається, мають певну особливість: змінювання стану в таких системах відбувається тільки в деякі моменти часу, а в усі інші моменти часу система не змінюється. Змінювання стану моделі спричиняється виникненням певної події у системі. Наприклад, подія «надходження деталі до технологічного процесу» спричиняє збільшення кількості деталей у системі, подія «виникнення поломки обладнання» спричиняє відправку поламаного обладнання до ремонту і т.д. Процес функціонування системи розглядається як послідовність подій, що відбуваються у моделі.

За принципом найближчої події модельний час просувається від моменту виникнення однієї події до моменту виникнення іншої, і після кожного просування часу реалізуються зміни стану моделі, відповідні до події, що виникла. Використання принципу найближчої події вимагає від дослідника побудови спеціальної процедури визначення моменту найближчої події, але при цьому він отримує виграш у затратах комп’ютерного часу, оскільки пропускається моделювання системи у моменти часу, коли події не відбуваються.

Розглянемо, наприклад, найпростішу модель обслуговування одним пристроєм потоку вимог (рисунок 2.3). В системі спостерігаються дві події: *V* - надходження вимоги, *W*– звільнення пристрою обслуговування. Представимо моменти часу, коли спостерігалась подія *V*, та моменти часу, коли спостерігалась подія *W*. Упорядкуємо моменти виконання подій *V* і *W* на вісі модельного часу *tм*. На першому кроці алгоритму імітації встановлюється момент модельного часу *tм1* та реалізується подія *V1*; на другому − момент модельного часу *tм2* тареалізуєтьсяподія*W1,* на третьому − *tм3* та реалізуються події *V2*і *W2* у відповідності з пріоритетом процесів, і т.д.

Для дискретних систем просування модельного часу за принципом найближчої події є найефективнішим способом.

Подія „надходження вимоги” *(V)* у реальному часі

Події *V* i *W* у модельному часі

Подія „закінчилось обслуговування ” *(W)* у реальному часі

*W1*

*W2*

*W3*

*t1(W)*

*t2(W)*

*t3(W)*

*t*

##### *V1 W1 V2 W2 V3 W3 V4*

##### *tM1 tM2 tM3 tM4 tM5 tM6*

*tМ*

*V1*

# *V3*

# *V4*

*t1(V)*

*t2(V)*

*t3(V)*

*t*

# *V2*

CREATE

DESPOSE

PROCESS

# Рисунок 2.3 - Упорядкування подій у модельному часі

###### *Принцип послідовного проведення об’єктів уздовж моделі*

За принципом послідовного проведення об’єктів уздовж моделі алгоритм просування часу не будується. Кожний об’єкт проводиться по моделі з моменту його надходження у модель до моменту виходу з моделі. Історія кожного проведення запам’ятовується, так що наступний об’єкт проводиться уздовж моделі з урахуванням історії попередніх проведень. Такий алгоритм імітації дозволяє моделювати систему тільки в моменти виникнення подій, не займаючись побудовою алгоритму просування модельного часу. Проте використовується він дуже рідко, оскільки часто призводить до складних заплутаних алгоритмів.

Способи побудови алгоритму просування стану моделі в залежності від часу

Процес функціонування імітаційної моделі може бути описаний з точки зору:

1. змінювання стану системи, що відбуваються в момент появи подій;
2. дій, які виконуються елементами системи;
3. процесу, який відбувається у системі.

У відповідності з означеними способами опису функціонування моделі існують три способи просування стану моделі в часі:

* орієнтований на події;
* орієнтований на дії;
* процесно-орієнтований.

*Спосіб, орієнтований на події*

При підході, орієнтованому на події, дослідник визначає і описує події, які виникають у моделі. Імітація здійснюється виконанням упорядкованої у часі послідовності логічно взаємозв’язаних подій.

Проілюструємо на прикладі моделі обслуговування одним пристроєм потоку вимог (див. рисунок 4.1). Стан системи описується станом пристрою обслуговування та станом черги. Змінювання стану системи відбувається у моменти, коли у систему надходить вимога і коли закінчився процес обслуговування вимоги у пристрої. Назвемо ці події подія «надходження» та подія «закінчилось обслуговування у пристрої».

Подія «надходження» складається з таких дій:

* якщо пристрій обслуговування в стані вільний установити пристрій у стан «зайнятий», запам’ятати момент виходу вимоги з пристрою в момент часу - поточний момент часу плюс тривалість обслуговування у пристрої;
* інакше
* якщо є вільне місце в черзі, то зайняти місце в черзі;
* інакше збільшити кількість не обслугованих вимог на одиницю;
* генерувати момент надходження наступної вимоги у СМО.

Подія «закінчилось обслуговування в пристрої» складається з таких дій:

* збільшити кількість обслугованих вимог на одиницю;
* якщо черга вимог не пуста, перемістити одну вимогу із черги у канал обслуговування, запам’ятати момент виходу вимоги з пристрою у момент часу - поточний момент часу плюс тривалість обслуговування у пристрої;
* інакше установити канал у стан «вільний» вільний стан, запам’ятати момент виходу вимоги з пристрою у момент часу, що більший за час моделювання (тобто у найближчий час вихід вимоги із пристрою не очікується).

Після виконання події «надходження» запам’ятовуються моменти виникнення наступних подій «надходження» та «закінчилось обслуговування у пристрої». З цих двох моментів часу вибирається найменший і запам’ятовується, яка саме подія відповідає цьому моменту. Модельний час просувається у визначений момент найближчої події і виконується відповідна йому подія.

При виконанні події «закінчилось обслуговування у пристрої» запам’ятовується момент наступної події «закінчилось обслуговування у пристрої» або запам’ятовується, що найближчим часом не очікується звільнення пристрою.

Таким чином, події виконуються в упорядкованій у часі послідовності, а модельний час просувається від одного моменту виникнення події до найближчого наступного.

Програма, яка здійснює імітацію моделі, що розглядається, складається з таких дій:

1. ввести початкові значення змінних - поточний часу, час моделювання, стан пристрою, момент надходження вимоги у систему, момент звільнення пристрою обслуговування;
2. доки поточний час менший за час моделювання
3. знайти найменший із моментів часу «момент надходження вимоги у систему» та «момент звільнення пристрою обслуговування» і запам’ятати, якій події він відповідає;
4. просунути поточний час у момент найближчої події;
5. виконати подію, яка відповідає моменту найближчої події.
6. вивести результати моделювання – кількість обслугованих вимог та кількість не обслугованих вимог протягом часу моделювання.

Наведемо лістинг програми мовою Java :

package simsimple;

public class SimSimple {

public static void main(String[] args) {

Model model = new Model(2,1,5);

model.simulate(1000);

}

}

public class Model {

private double tnext;

private double tcurr;

private double t0, t1;

private double delayCreate, delayProcess;

private int numCreate, numProcess, failure;

private int state, maxqueue, queue;

private int nextEvent;

public Model(double delay0, double delay1){

delayCreate = delay0;

delayProcess = delay1;

tnext=0.0;

tcurr = tnext;

t0=tcurr; t1=Double.MAX\_VALUE;

maxqueue=0;

}

public Model(double delay0, double delay1, int maxQ){

delayCreate = delay0;

delayProcess = delay1;

tnext=0.0;

tcurr = tnext;

t0=tcurr; t1=Double.MAX\_VALUE;

maxqueue=maxQ;

}

public void simulate(double timeModeling){

while(tcurr<timeModeling){

tnext = t0;

nextEvent = 0;

if(t1<tnext){

tnext = t1;

nextEvent = 1;

}

tcurr = tnext;

switch(nextEvent){

case 0: event0();

break;

case 1: event1();

}

printInfo();

}

printStatistic();

}

public void printStatistic(){

System.out.println(" numCreate= " + numCreate+" numProcess = "+numProcess+" failure = "+failure);

}

public void printInfo(){

System.out.println(" t= " + tcurr+" state = "+state+" queue = "+queue);

}

public void event0(){

t0 = tcurr+getDelayOfCreate();

numCreate++;

if(state==0){

state = 1;

t1 = tcurr+getDelayOfProcess();

} else {

if(queue<maxqueue)

queue++;

else

failure++;

}

}

public void event1(){

t1 = Double.MAX\_VALUE;

state=0;

if(queue>0){

queue--;

state=1;

t1 = tcurr+getDelayOfProcess();

}

numProcess++;

}

private double getDelayOfCreate() {

return FunRand.Exp(delayCreate);

}

private double getDelayOfProcess() {

return FunRand.Exp(delayProcess);

}

}

public class FunRand {

/\*\*

\* Generates a random value according to an exponential distribution

\*

\* @param timeMean mean value

\* @return a random value according to an exponential distribution

\*/

public static double Exp(double timeMean) {

double a = 0;

while (a == 0) {

a = Math.random();

}

a = -timeMean \* Math.log(a);

return a;

}

/\*\*

\* Generates a random value according to a uniform distribution

\*

\* @param timeMin

\* @param timeMax

\* @return a random value according to a uniform distribution

\*/

public static double Unif(double timeMin, double timeMax) {

double a = 0;

while (a == 0) {

a = Math.random();

}

a = timeMin + a \* (timeMax - timeMin);

return a;

}

/\*\*

\* Generates a random value according to a normal (Gauss) distribution

\*

\* @param timeMean

\* @param timeDeviation

\* @return a random value according to a normal (Gauss) distribution

\*/

public static double Norm(double timeMean, double timeDeviation) {

double a;

Random r = new Random();

a = timeMean + timeDeviation \* r.nextGaussian();

return a;

}

}