**Лабораторна робота №3**

з курсу «Сучасні операційні системи»

Виконала:

студентка 3 курсу ФІОТ

групи ІС-73

Мигуля Ірина

Перевірив:

Дифучин А.Ю.

**Варіант: 12+1=13**

**MFQS (multilevel feedback queue sheduling )**

Звичайна багаторівнева черга не допускає переміщення заявки між чергами. Багаторівнева черга з оберненими зв'язками передбачає, що заявки при певних умовах можуть переміщатися між чергами.

Процеси початково потрапляють в чергу 0, де кожному із них надається квант часу, рівний 8 мс. Ті процеси, котрі не встигли виконатися протягом цього часу, переміщаються в чергу 1. Процеси із черги 1 починають оброблятися тільки тоді, коли черга 0 стає пустою. Ті процеси, котрі не виконались в черзі 1 (q=16 мс) переміщаються в чергу 2. Процеси із черги 2 будуть оброблятися тільки в тому випадку, якщо стають пустими черги 0 та 1.

**Лістинг програми**

**public** **class** Request {

**private** **int** weight;

**private** **int** priority;

**private** **int** processed;

**private** **int** inTime;

**private** **int** outTime;

**public** Request(**int** weight, **int** priority, **int** inTime) {

**this**.weight = weight;

**this**.priority = priority;

**this**.inTime = inTime;

processed=0;

}

**public** **int** getWeight() {

**return** weight;

}

**public** **int** getPriority() {

**return** priority;

}

**public** **void** incProcessed() {

processed++;

}

**public** **int** getProcessed() {

**return** processed;

}

**public** **void** setInTime(**int** inTime) {

**this**.inTime = inTime;

}

**public** **void** setOutTime(**int** outTime) {

**this**.outTime = outTime;

}

**public** **int** getInTime() {

**return** inTime;

}

**public** **int** getOutTime() {

**return** outTime;

}

}

public class Queue {

private int n;

private Request[] queue;

private int defaultPriority;

public Queue(int defaultPriority, int n) {

this.defaultPriority=defaultPriority;

this.n=n;

queue = new Request[n];

}

public void addRequest(Request request) {

int i=n;

do {

i--;

}

while ((i>0)&(queue[i]!=null));

if ((i==0)&(queue[i]!=null)) {

ExtendQueue(2\*n);

i=n/2-1;

}

queue[i]=request;

}

public String getQueue() {

String res = "";

for (int i=n-1;i>=0;i--) {

if (queue[i]!=null) {

res=res+queue[i].getProcessed()+"/"+queue[i].getWeight()+ " ";

}

}

return res;

}

public boolean isEmpty() {

if (queue[n-1]==null) {

return true;

}

else

return false;

}

public Request ExtractRequest() {

Request request;

request=queue[n-1];

for (int i=n-2;i>=0; i--) {

queue[i+1]=queue[i];

}

queue[0]=null;

return request;

}

public void ExtendQueue(int n) {

Request[] newQueue = new Request[n];

for (int i=this.n-1; i>=0;i--) {

newQueue[i+n-this.n]=queue[i];

}

queue=newQueue;

this.n=n;

}

}

public class QueueBlock {

private Queue[] queueBlock;

private int number;

private int minPriority;

public QueueBlock(int n, int minPriority) {

number = n+1;

this.minPriority = minPriority;

queueBlock = new Queue[number];

BlockInitialization();

}

private void BlockInitialization() {

for (int i=0;i<number;i++) {

queueBlock[i] = new Queue(i+minPriority,1);

}

}

public void addRequestWithFilter(Request request) {

queueBlock[request.getPriority()-minPriority].addRequest(request);

}

public void addRequestDirectly(Request request, int queueNumber) {

queueBlock[queueNumber].addRequest(request);

}

public String getBlock() {

String res = "";

for (int i=0;i<number;i++) {

res=res+("Приоритет: "+(i+minPriority)+" | ");

res=res+queueBlock[i].getQueue()+"\n";

}

return res;

}

public Request ExtractMaxPriorityRequest() {

int i = -1;

do {

i++;

}

while ((i<number-1)&(queueBlock[i].isEmpty()));

if (queueBlock[number-1]!=null) {

if (i!=number-1)

return queueBlock[i].ExtractRequest();

else

return queueBlock[number-1].ExtractRequest();

}

else

return null;

}

public int getMinPriority() {

return minPriority;

}

public int getQueueNumber() {

return number;

}

}

public class Resource {

private Request currentRequest;

private int processingTime;

private int quant;

private int idleTime = 0;

private boolean occupied = false;

private QueueBlock queue;

private int counter = 0;

private Form form;

private int[] priorities;

private short[] number;

public Resource(Form form, QueueBlock queue, int quant) {

this.form = form;

this.queue = queue;

this.quant = quant;

priorities = new int[queue.getMinPriority()+queue.getQueueNumber()+1];

number = new short[queue.getMinPriority()+queue.getQueueNumber()+1];

for (int i=0;i<priorities.length;i++) {

priorities[i]=0;

number[i] = 0;

}

}

public void setRequest(Request request) {

currentRequest = request;

processingTime = 0;

if (request!=null) {

occupied = true;

}

else

occupied = false;

}

public void processOneTact() {

if (occupied) {

currentRequest.incProcessed();

processingTime++;

if (currentRequest.getProcessed()>=currentRequest.getWeight()) {

currentRequest.setOutTime(form.getTime());

form.addWaitingTime(currentRequest.getOutTime()-currentRequest.getInTime()-currentRequest.getWeight());

int waitingTime=(form.getTime()-currentRequest.getInTime()-currentRequest.getWeight());

priorities[currentRequest.getPriority()]+=waitingTime;

number[currentRequest.getPriority()]++;

setRequest(queue.ExtractMaxPriorityRequest());

form.setRequestsNumber(form.getRequestsNumber()-1);

counter++;

}

if (processingTime>=quant) {

queue.addRequestDirectly(currentRequest, currentRequest.getPriority()-queue.getMinPriority());

setRequest(queue.ExtractMaxPriorityRequest());

}

}

else {

idleTime++;

setRequest(queue.ExtractMaxPriorityRequest());

}

}

public Request getCurrentRequest() {

return currentRequest;

}

public int getQuant() {

return quant;

}

public int getProcessingTime() {

return processingTime;

}

public int getIdleTime() {

return idleTime;

}

public void setIdleTime(int time) {

idleTime = time;

}

public int getProcessedNumber() {

return counter;

}

public void setQueue(QueueBlock queue) {

this.queue=queue;

}

public int[] getPriorityArray() {

return priorities;

}

public short[] getProcessedNumberWithPriorities() {

return number;

}

}

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** javax.swing.\*;

**public** **class** Graph **extends** JFrame{

**protected** **short**[] value;

**protected** Form form;

**protected** String x,y;

**public** Graph(Form form, **short**[] value, String x, String y) {

**super**("График");

**this**.form=form;

setSize(510, 600);

setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);

setResizable(**false**);

**this**.value = value;

**this**.x=x;

**this**.y=y;

setVisible(**true**);

}

**public** **void** paint(Graphics g) {

g.setColor(Color.*BLACK*);

DrawAxises(g);

g.drawOval(50, 580-value[1]\*2,1,1);

**if** ((580-value[1]\*2)<30) {

g.drawString(String.*valueOf*(value[1]),70,60);

}

g.drawString(String.*valueOf*(value[1]),60,580-value[1]\*2);

**for** (**int** i=form.getIMin()+1; i<=form.getIMax(); i++) {

g.setColor(Color.*BLACK*);

**if** ((value[i]!=32767)) {

g.drawOval(40+i\*10, 580-value[i]\*2, 1, 1);

}

g.setColor(Color.*RED*);

**if** (value[i-1]!=32767) {

g.drawLine(40+(i-1)\*10, 580-value[i-1]\*2, 40+(i)\*10, 580-value[i]\*2);

}

}

}

**public** **void** DrawAxises(Graphics g) {

g.drawString(y,10,40);

g.drawString(x,410,593);

g.drawLine(40, 40, 40, 580);

g.drawLine(40, 580, 480, 580);

**for** (**int** i=0; i<27;i++) {

g.drawString(String.*valueOf*(i\*10),15,580-i\*20);

g.drawLine(38,580-i\*20,42,580-i\*20);

}

**for** (**int** i=0; i<9; i++) {

g.drawString(String.*valueOf*(i\*4),40+i\*40,593);

g.drawLine(40+i\*40, 582, 40+i\*40, 578);

}

**for** (**int** i=9; i<12;i++) {

g.drawString(String.*valueOf*(i\*4),40+i\*40, 578);

g.drawLine(40+i\*40, 582, 40+i\*40, 578);

}

}

}

**import** java.awt.Color;

**import** java.awt.Graphics;

**public** **class** Graph2 **extends** Graph {

**public** Graph2(Form form,**short**[] value, String x, String y) {

**super**(form,value,x,y);

setSize(510,300);

}

**public** **void** paint(Graphics g) {

g.setColor(Color.*BLACK*);

DrawAxises(g);

g.drawOval(50, 280-value[1]\*2,1,1);

g.drawString(String.*valueOf*(value[form.getIMax()-1]),40+(form.getIMax()-1)\*10,280-value[form.getIMax()-1]\*2);

**for** (**int** i=form.getIMin()+1; i<form.getIMax(); i++) {

g.setColor(Color.*BLACK*);

g.drawOval(40+i\*10, 280-value[i]\*2, 1, 1);

g.setColor(Color.*RED*);

g.drawLine(40+(i-1)\*10, 280-value[i-1]\*2, 40+(i)\*10, 280-value[i]\*2);

}

}

**public** **void** DrawAxises(Graphics g) {

g.drawString(y,10,40);

g.drawString(x,410,293);

g.drawLine(40, 40, 40, 280);

g.drawLine(40, 280, 480, 280);

**for** (**int** i=0; i<11;i++) {

g.drawString(String.*valueOf*(i\*10),15,280-i\*20);

g.drawLine(38,280-i\*20,42,280-i\*20);

}

**for** (**int** i=0; i<9; i++) {

g.drawString(String.*valueOf*(i\*4),40+i\*40,293);

g.drawLine(40+i\*40, 282, 40+i\*40, 278);

}

**for** (**int** i=9; i<12;i++) {

g.drawString(String.*valueOf*(i\*4),40+i\*40, 278);

g.drawLine(40+i\*40, 282, 40+i\*40, 278);

}

}

}

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** javax.swing.JFrame;

**public** **class** Graph3 **extends** JFrame {

**protected** **int** value[];

**protected** Form form;

**protected** String x,y;

**public** Graph3(Form form, **int**[] value, String x, String y) {

**super**("График зависимости ср. времени ожидания от приоритета заявок");

**this**.form=form;

**this**.x=x;

**this**.y=y;

**this**.value=value;

setSize(700, 610);

setVisible(**true**);

}

**public** **void** paint(Graphics g) {

g.setColor(Color.*BLACK*);

DrawAxises(g);

g.drawOval(40+form.getPrMin()\*20, 580-value[form.getPrMin()],1,1);

g.drawString(String.*valueOf*(value[form.getPrMin()]),38+20, 578-value[form.getPrMin()]);

**for** (**int** i=form.getPrMin()+1; i<=form.getPrMax(); i++) {

g.setColor(Color.*BLACK*);

g.drawOval(40+i\*20, 580-value[i], 1, 1);

g.setColor(Color.*RED*);

g.drawLine(40+(i-1)\*20, 580-value[i-1], 40+(i)\*20, 580-value[i]);

g.setColor(Color.*BLACK*);

g.drawString(String.*valueOf*(value[i]),38+i\*20, 578-value[i]);

}

}

**public** **void** DrawAxises(Graphics g) {

g.drawString(y,10,40);

g.drawString(x,610,593);

g.drawLine(40, 40, 40, 580);

g.drawLine(40, 580, 680, 580);

**for** (**int** i=0; i<27;i++) {

g.drawString(String.*valueOf*(i\*20),15,580-i\*20);

g.drawLine(38,580-i\*20,42,580-i\*20);

}

**for** (**int** i=0; i<13; i++) {

g.drawString(String.*valueOf*(2\*i),40+i\*40,593);

g.drawLine(40+i\*40, 582, 40+i\*40, 578);

}

**for** (**int** i=13; i<16;i++) {

g.drawString(String.*valueOf*(2\*i),40+i\*40, 578);

g.drawLine(40+i\*40, 582, 40+i\*40, 578);

}

}

}

**import** java.awt.Color;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.\*;

**import** javax.swing.border.LineBorder;

**public** **class** Form **extends** JFrame {

**private** JLabel iminLabel = **new** JLabel("Мин. t поступления: ");

**private** JLabel imaxLabel = **new** JLabel("Макс. t поступления: ");

**private** JTextField imin = **new** JTextField(5);

**private** JTextField imax = **new** JTextField(5);

**private** JLabel wminLabel = **new** JLabel("Мин. вес: ");

**private** JLabel wmaxLabel = **new** JLabel("Макс. вес: ");

**private** JTextField wmin = **new** JTextField(5);

**private** JTextField wmax = **new** JTextField(5);

**private** JLabel prminLabel = **new** JLabel("Мин. приоритет: ");

**private** JLabel prmaxLabel = **new** JLabel("Макс. приоритет: ");

**private** JTextField prmin = **new** JTextField(5);

**private** JTextField prmax = **new** JTextField(5);

**private** JPanel iPanel = **new** JPanel();

**private** JPanel wPanel = **new** JPanel();

**private** JPanel prPanel = **new** JPanel();

**private** JPanel fieldsPanel = **new** JPanel();

**private** JPanel buttonsPanel = **new** JPanel();

**private** JPanel textPanel = **new** JPanel();

**private** JTextArea area = **new** JTextArea(20,50);

**private** JScrollPane scroll = **new** JScrollPane(area);

**private** JButton tact = **new** JButton("Выполнить такт");

**private** JButton graph1 = **new** JButton("График Тож(I)");

**private** JButton graph2 = **new** JButton("График idle(I)");

**private** JButton graph3 = **new** JButton("График Тож(Pr)");

**private** JButton graphs = **new** JButton("Построение графиков");

**private** JButton init = **new** JButton("Задать значения");

**private** JTextField quant = **new** JTextField(5);

**private** JLabel quantLabel = **new** JLabel("Квант времени: ");

**private** JPanel quantPanel = **new** JPanel();

**private** JButton values = **new** JButton("Задать новые значения");

**private** JLabel nLabel = **new** JLabel("Кол-во тактов: ");

**private** JTextField nField = **new** JTextField(5);

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

**private** **int** iminValue;

**private** **int** imaxValue;

**private** **int** wminValue;

**private** **int** wmaxValue;

**private** **int** prminValue;

**private** **int** prmaxValue;

**private** **int** time = 0;

**private** **int** waitingTime;

**private** **int** quantValue;

**private** **int** waitingTime\_all=0;

**private** **int** requestsNum = 0;

**private** QueueBlock queue;

**private** Resource resource;

**public** Form() {

**super**("Lab3");

setSize(600,500);

setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);

Interface();

setVisible(**true**);

init.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

Init();

setFieldsEditable(**false**);

init.setVisible(**false**);

tact.setVisible(**true**);

graphs.setVisible(**true**);

values.setVisible(**true**);

}

});

tact.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

ExecuteOneTact(queue, resource);

Visualisation(queue, resource);

}

});

values.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

values.setVisible(**false**);

setFieldsEditable(**true**);

graph1.setVisible(**false**);

graph2.setVisible(**false**);

graph3.setVisible(**false**);

graphs.setVisible(**false**);

tact.setVisible(**false**);

init.setVisible(**true**);

}

});

graphs.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

tact.setVisible(**false**);

nField.setEditable(**true**);

graphs.setVisible(**false**);

graph1.setVisible(**true**);

graph2.setVisible(**true**);

graph3.setVisible(**true**);

values.setVisible(**true**);

}

});

graph1.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

**int** min = iminValue;

**int** max = imaxValue;

**short**[] array = **new** **short**[imaxValue+iminValue+1];

**for** (**int** i=1;i<=max;i++) {

imaxValue=i;

iminValue=i;

QueueBlock testBlock = **new** QueueBlock(prmaxValue-prminValue,prminValue);

resource.setQueue(testBlock);

resource.setRequest(**null**);

**for** (**short** j=0;j<Integer.*parseInt*(nField.getText());j++) {

ExecuteOneTact(testBlock, resource);

}

array[iminValue]=(**short**)(waitingTime\_all/resource.getProcessedNumber());

waitingTime\_all=0;

}

iminValue=min;

imaxValue=max;

resource.setQueue(queue);

Init();

**new** Graph(Form.**this**, array, "Тпоступления","Tожидания");

}

});

graph2.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

**int** min = iminValue;

**int** max = imaxValue;

**short**[] array = **new** **short**[imaxValue+iminValue+1];

**for** (**int** i=1;i<max;i++) {

imaxValue=i;

iminValue=i;

QueueBlock testBlock = **new** QueueBlock(prmaxValue-prminValue,prminValue);

resource.setQueue(testBlock);

resource.setRequest(**null**);

**for** (**short** j=0;j<Integer.*parseInt*(nField.getText());j++) {

ExecuteOneTact(testBlock, resource);

}

array[iminValue]=(**short**)(((**float**)resource.getIdleTime()/time)\*100);

time=0;

resource.setIdleTime(0);

}

iminValue=min;

imaxValue=max;

resource.setQueue(queue);

Init();

**new** Graph2(Form.**this**, array, "Тпоступления","% простоя");

}

});

graph3.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

Init();

QueueBlock testBlock = **new** QueueBlock(prmaxValue-prminValue,prminValue);

resource.setQueue(testBlock);

resource.setRequest(**null**);

**for** (**short** j=0;j<Integer.*parseInt*(nField.getText());j++) {

ExecuteOneTact(testBlock, resource );

}

**int**[] array = resource.getPriorityArray();

**short**[] numbers = resource.getProcessedNumberWithPriorities();

Request r;

**while** ((r = testBlock.ExtractMaxPriorityRequest())!=**null**) {

numbers[r.getPriority()]++;

array[r.getPriority()]+=getTime()-r.getInTime()-r.getProcessed();

}

**for** (**int** i=1;i<array.length; i++) {

**if** (numbers[i]!=0) {

array[i]/=numbers[i];

}

}

resource.setQueue(queue);

**new** Graph3(Form.**this**, array, "Приоритет","Тожидания");

}

});

}

**public** **void** Interface() {

getContentPane().setLayout(**new** BoxLayout(getContentPane(), BoxLayout.*Y\_AXIS*));

getContentPane().add(fieldsPanel);

getContentPane().add(buttonsPanel);

getContentPane().add(textPanel);

fieldsPanel.setBorder(**new** LineBorder(Color.*BLACK*));

fieldsPanel.add(iPanel);

fieldsPanel.add(wPanel);

fieldsPanel.add(prPanel);

fieldsPanel.add(quantPanel);

iPanel.setLayout(**new** BoxLayout(iPanel, BoxLayout.*Y\_AXIS*));

wPanel.setLayout(**new** BoxLayout(wPanel, BoxLayout.*Y\_AXIS*));

prPanel.setLayout(**new** BoxLayout(prPanel, BoxLayout.*Y\_AXIS*));

quantPanel.setLayout(**new** BoxLayout(quantPanel, BoxLayout.*Y\_AXIS*));

iPanel.add(iminLabel);

iPanel.add(imin);

iPanel.add(imaxLabel);

iPanel.add(imax);

wPanel.add(wminLabel);

wPanel.add(wmin);

wPanel.add(wmaxLabel);

wPanel.add(wmax);

prPanel.add(prminLabel);

prPanel.add(prmin);

prPanel.add(prmaxLabel);

prPanel.add(prmax);

quantPanel.add(quantLabel);

quantPanel.add(quant);

quantPanel.add(nLabel);

quantPanel.add(nField);

nField.setText("30000");

nField.setEditable(**false**);

buttonsPanel.setBorder(**new** LineBorder(Color.*BLUE*));

buttonsPanel.add(init);

tact.setVisible(**false**);

graphs.setVisible(**false**);

values.setVisible(**false**);

graph1.setVisible(**false**);

graph2.setVisible(**false**);

graph3.setVisible(**false**);

buttonsPanel.add(tact);

buttonsPanel.add(graphs);

tact.setToolTipText("Выполнить один такт");

graph1.setToolTipText("Построить график зависимости среднего времени ожидания от интенсивности поступления заявок");

graph2.setToolTipText("Построить график зависимости процента простоя ресурса от интенсивности поступления заявок");

graph3.setToolTipText("Построить график зависимости среднего времени ожидания от приоритета");

buttonsPanel.add(graph1);

buttonsPanel.add(graph2);

buttonsPanel.add(graph3);

buttonsPanel.add(values);

textPanel.add(scroll);

area.setEditable(**false**);

setVisible(**true**);

}

**private** **void** Init() {

time = 0;

requestsNum = 0;

iminValue = Integer.*parseInt*(imin.getText());

imaxValue = Integer.*parseInt*(imax.getText());

wminValue = Integer.*parseInt*(wmin.getText());

wmaxValue = Integer.*parseInt*(wmax.getText());

prminValue = Integer.*parseInt*(prmin.getText());

prmaxValue = Integer.*parseInt*(prmax.getText());

quantValue = Integer.*parseInt*(quant.getText());

queue = **new** QueueBlock(prmaxValue-prminValue, prminValue);

waitingTime = iminValue+(**int**)(Math.*round*(Math.*random*()\*(imaxValue-iminValue)));

resource = **new** Resource(**this**, queue, quantValue);

resource.setRequest(**null**);

}

**public** **void** ExecuteOneTact(QueueBlock queue, Resource resource) {

resource.processOneTact();

time++;

waitingTime--;

**if** (waitingTime == 0) {

queue.addRequestWithFilter(**new** Request(wminValue+(**int**)(Math.*round*(Math.*random*()\*(wmaxValue-wminValue))),prminValue+(**int**)(Math.*round*(Math.*random*()\*(prmaxValue-prminValue))),time));

requestsNum+=1;

waitingTime = iminValue+(**int**)(Math.*round*(Math.*random*()\*(imaxValue-iminValue)));

}

}

**public** **void** Visualisation(QueueBlock queue, Resource resource) {

area.setText(queue.getBlock());

**if** (resource.getCurrentRequest()!=**null**) {

area.append("Обработка текущей заявки: "+resource.getCurrentRequest().getProcessed()+"/"+resource.getCurrentRequest().getWeight()+ "(Приоритет: "+ resource.getCurrentRequest().getPriority()+") \n");

area.append("Квант: "+String.*valueOf*(resource.getQuant())+ "\n");

area.append("Время пребывания текущей заявки на ресурсе "+String.*valueOf*(resource.getProcessingTime())+"\n");

}

**else** {

area.append("Ресурс простаивает. Общее время простоя: "+String.*valueOf*(resource.getIdleTime())+ "\n");

area.append("Процент простоя: "+ ((**float**)resource.getIdleTime()/time)\*100 +"\n");

}

area.append("Общее время работы системы: "+getTime()+" \n");

area.append("Текущее кол-во заявок в системе: "+getRequestsNumber()+"\n");

area.append("Заявок обработано: "+resource.getProcessedNumber()+"\n");

}

**public** **void** addWaitingTime(**int** time) {

waitingTime\_all+=time;

}

**public** **int** getRequestsNumber() {

**return** requestsNum;

}

**public** **void** setRequestsNumber(**int** num) {

requestsNum = num;

}

**public** **int** getTime() {

**return** time;

}

**public** **int** getIMin() {

**return** iminValue;

}

**public** **int** getIMax() {

**return** imaxValue;

}

**public** **void** setFieldsEditable(**boolean** value) {

imin.setEditable(value);

imax.setEditable(value);

wmin.setEditable(value);

wmax.setEditable(value);

prmin.setEditable(value);

prmax.setEditable(value);

quant.setEditable(value);

}

**public** **int** getPrMin() {

**return** prminValue;

}

**public** **int** getPrMax() {

**return** prmaxValue;

}

**public** **static** **void** main(String[] args) {

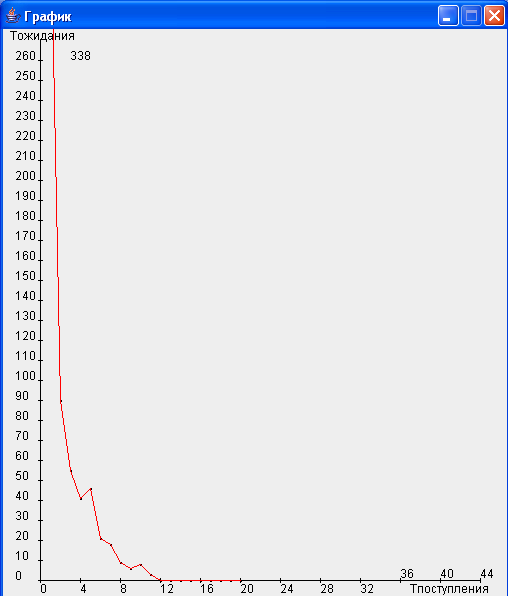
**new** Form();

}

}

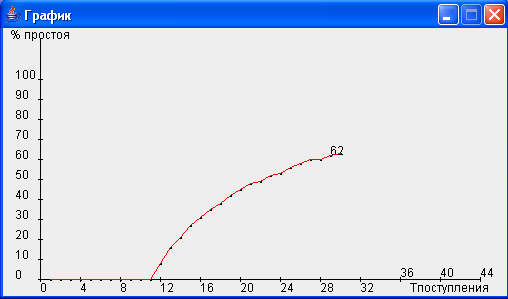
**Графіки**

* Графік залежності середнього часу очікування від середнього часу очікування надходження заявки.



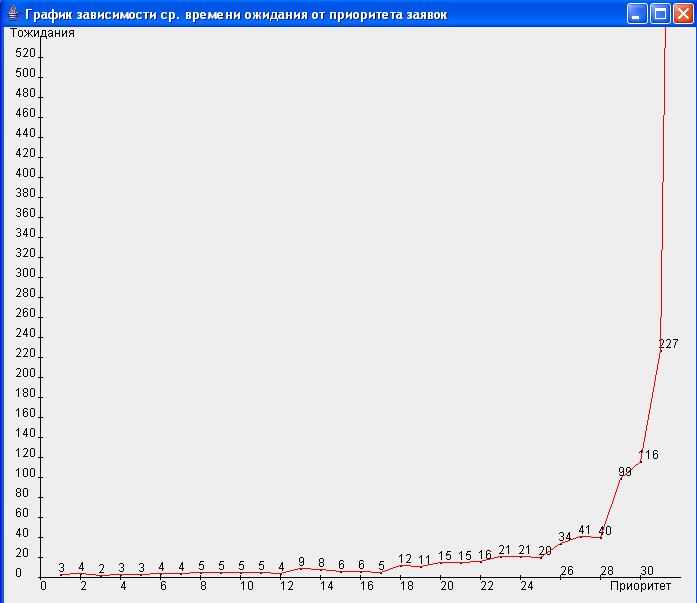
Графік спадає, тому що чим довші проміжки між надходженням нових заявок, тим менше їх знаходиться в чергах, і тим швидше заявки дочекаються обробки.

* Графік залежності відсотка простоя ресурсу від середнього часу очікування надходження заявки.



Графік зростає, оскільки чим довше проміжки між надходженнями нових заявок, тим менше навантажений ресурс, і тим частіше він не має заявок для обробки і простоює.

* Графік залежності середнього часу очікування від пріоритету заявок



Графік зростає, оскільки чим нижчий пріоритет у заявки, тим довше ресурс надає перевагу їй, заявки з більш високим пріоритетом (найвищий 1) і тим довше заявка очікує обробки.