НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
“КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ”

КУРСОВАЯ РАБОТА

по дисциплине:“Операционные системы”

Выполнил:
Федосеев Тарас
Студент 4 курса ФИВТ
Группы ИВ-73

Киев 2010

Техническое задание

Техническое задание

Разработать пространственный планировщик для распределения заданий в многопроцессорной системе и исследовать его работу

Топология :

Полносвязная система - неоднородная

Пояснительная записка

Содержание

Введение

Общая задача планирования включает в себя задачу присвоения заданий приложения доступным подходящим процессорам и задачу упорядочивания времени запуска заданий на каждом ресурсе. Когда такие параметры приложения как время, необходимое для выполнения каждого задания в приложении, объем передаваемых данных между разными приложениями, а также зависимости между заданиями в приложении, известны заранее, то речь идёт о статическом планировании.

В общем виде при решении задач статического планирования заданий приложение обычно представляется направленным ациклическим графом (DAG), в котором вершины представляют задания в приложении, а рёбра представляют связи по данным между заданиями. Вес каждой вершины представляет ожидаемое время выполнения данного задания, а вес каждого ребра – ожидаемое время обмена информацией между заданиями. Цель статического планирования – присвоение заданий процессорам, а также упорядочивание времён их запуска таким образом, чтобы соблюдались зависимости по данным, и при этом достигалось наименьшее общее время выполнения приложения. Задача планирования заданий NP-полная в общем случае , также как и в некоторых частных случаях.

Полносвязная топология (рис. 4.2) соответствует сети, в которой каждый компьютер непосредственно связан со всеми остальными. Несмотря на логическую простоту, это вариант громоздкий и неэффективный. Действительно, каждый компьютер в сети должен иметь большое количество коммуникационных портов, достаточное для связи с каждым из остальных компьютеров. Для каждой пары компьютеров должна быть выделена отдельная физическая линия связи. (В некоторых случаях даже две, если невозможно использование этой линии для двусторонней передачи.) Полносвязные топологии в крупных сетях применяются редко, так как для связи N узлов

требуется N(N-1)/2 физических дуплексных линий связи, т.е. имеет место квадратичная зависимость. Чаще этот вид топологии используется в многомашинных комплексах или в сетях, объединяющих небольшое количество компьютеров.

1.Описание алгоритма планирования (HEFT)

В данной модели был реализован алгоритм HEFT как одни из самых высокоприозводительных алгоритмов планирования для неоднородных систем, а также один из разновидностей генетических алгоритмов.

Перед описанием алгоритма необходимо ввести некоторые понятия.

Задания в списке упорядочиваются на основании прямых и обратных рангов. Прямой ранг вычисляется по следующей формуле:

$$rank_u(n_i) = w_i + \max_{n_i \in succ(n_i)} (c_{i,j} + rank_u(n_j))$$

где $succ(n_i)$ - множество непосредственных наследников задания n_i, $c_{i,j}$ – время межпроцессорного взаимодействия для ребра (i, j), w_i – время выполнения задания. Фактически, $rank_u(n_i)$ - длина критического пути от задания n_i до конечной вершины.

Аналогично, обратный ранг рекурсивно вычисляется следующим образом:

$$rank_d(n_i) = \max_{n_i \in pred(n_i)} (c_{i,j} + w_i + rank_d(n_j)),$$

где $pred(n_i)$ - множество непосредственных предшественников задания n_i. Фактически, $rank_d(n_i)$ - самый большой путь от начальной вершины до текущей, исключая вес текущей вершины.

Теперь, опишем сам HEFT-алгоритм. Вкратце, его можно задать следующей последовательностью действий:

- Вычислить $rank_u(n_i)$ для всех заданий графа, обходя граф вверх, начиная с конечной вершины.
- Отсортировать список задач в порядке уменьшения $rank_u(n_i)$
- while в списке есть нераспланированные задачи do

- Выбрать первую задачу из списка
- for каждый процессор из доступного набора процессоров do
- Вычислить $EFT(n_i, p_k)$ на основе политики *включения заданий*
- Назначить задание n_i такому процессору p_j который
  - минимизирует EFT задачи n_i.
- end while

Алгоритм разделён на две основные фазы. Первая фаза – фаза назначения приоритетов заданиям, вторая фаза – фаза выбора «лучшего» процессора для каждой задачи, процессора, который минимизирует время завершения выполнения этой задачи.

Фаза назначения приоритетов

Во время этой фазы задачам назначаются приоритеты, в соответствии с которыми упорядочивается задачи в списке планирования. В базовой версии алгоритма в качестве приоритета выбирается величина $rank_u(n_i)$, но впрочем, существуют и другие политики назначения приоритетов (например – Ранг вершины = $rank_u(n_i) + rank_d(n_i)$). При сортировке в базовой версии алгоритма в случае одинаковых приоритетов задача для планирования выбирается произвольно. Существуют и альтернативные политики выбора задачи в случае равных приоритетов, однако поскольку они увеличивают временную сложность алгоритма, автор алгоритма использует политику произвольного выбора.

Фаза выбора процессора

В большинстве алгоритмов задания назначаются на процессоры после самого последнего задания, уже назначенного на этот процессор. Однако HEFT использует политику «включения» заданий. То есть при назначении задания на процессор алгоритм ищет в расписании процессора «окно» достаточной ширины и задача включается в найденное окно, если на тот момент выполняются условия предшествования. Для уменьшения временной сложности алгоритма поиск окна начинается не с начала расписания процессора, а с того

момента, когда все непосредственные предшественники вершины завершили своё выполнение. Поиск продолжается до того момента, пока не будет найдено «окно», в которое «помещается» текущая вершина. Если такое окно не найдено, то вершина помещается в конец расписания для соответствующего процессора. Временная сложность алгоритма HEFT равна $O(e*p)$ для e рёбер и p процессоров. Для плотных графов, в которых количество рёбер пропорционально $O(v^2)$ (v – количество заданий), временная сложность алгоритма – $O(v^2*p)$.

2.Интерфейс программы

Данная программа является графическим редактором графа задачи. Вид главного окна программы представлен на рисунке 1.

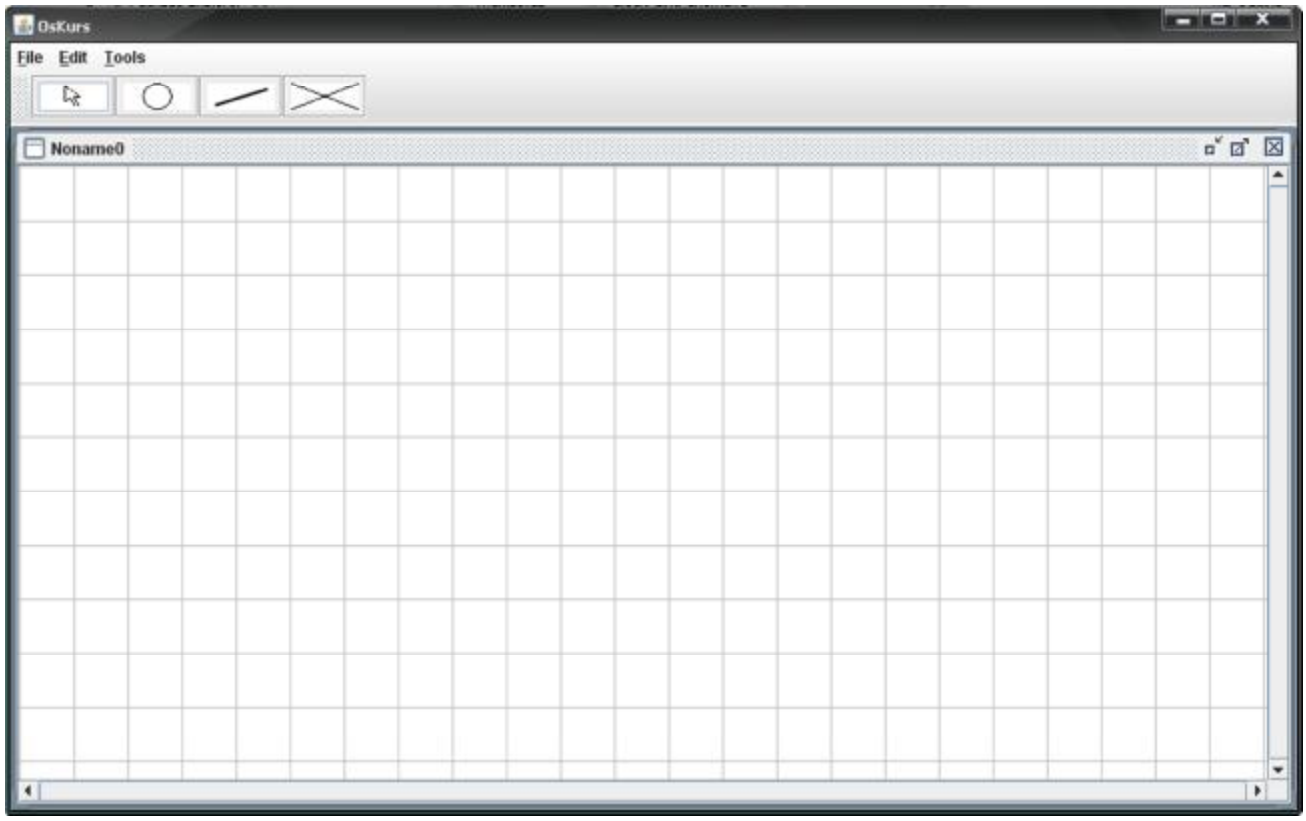


Рис 1

Для создания вершины можно выбрать на панели быстрого доступа кнопку с изображением круга(нажатием левой кнопки мыши) , затем переведя курсор на поле набора нажать левую кнопку. После этого на поле набора графа мы увидим всплывающее окно, в котором необходимо будет ввести вес вершины рис 2.

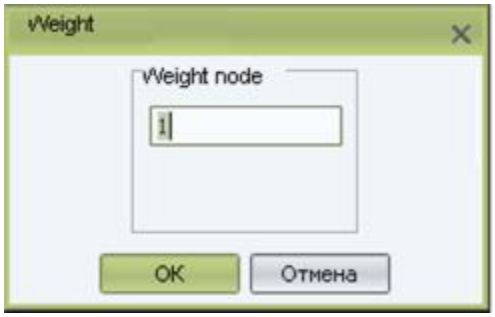


Рис 2.

После того как мы укажем необходимый вес то на поле набора графа появится вершина, номер вершины выбирается последовательно, а вес вершины тот ,который мы указали ранее в сплывающем окне. Для добавления ребра графа необходимо выбрать кнопку с изображением диагональной линии на панели быстрого доступа. Затем выбрать две вершины ,которые данное ребро будет соединять. После этого на поле набора графа мы увидим всплывающее окно, в котором необходимо будет ввести вес связи графа рис 3.

Рис 3.

После того как мы укажем необходимый вес , на поле набора графа появится связь между двумя выбранными вершинами, а вес связи тот ,который мы указали ранее в сплывающем окне. Выбрав кнопку с изображением стрелки мы можем (нажимая левую кнопку мыши) выбирать необходимую вершину и перемещать ее по полю набора графа. Связи перемещаются автоматически за вершинами. Для удаление вершины или связи необходимо выбрать кнопку с

изображением крестика. Затем при нажатии на связь или вершину на поле набора графа, она будет удалена.

Меню «File»: New , Open , Save.

Меню «Tools»: System setup, Run

С помощью «File» мы можем создать новый граф, открыть старый, или сохранить созданный.

С помощью «Tools» мы определяем конфигурацию системы (рис .4) и запускаем моделирование.

Рис 4.

3.Результаты моделирования

При моделировании был использован граф изображенный на рис 5.

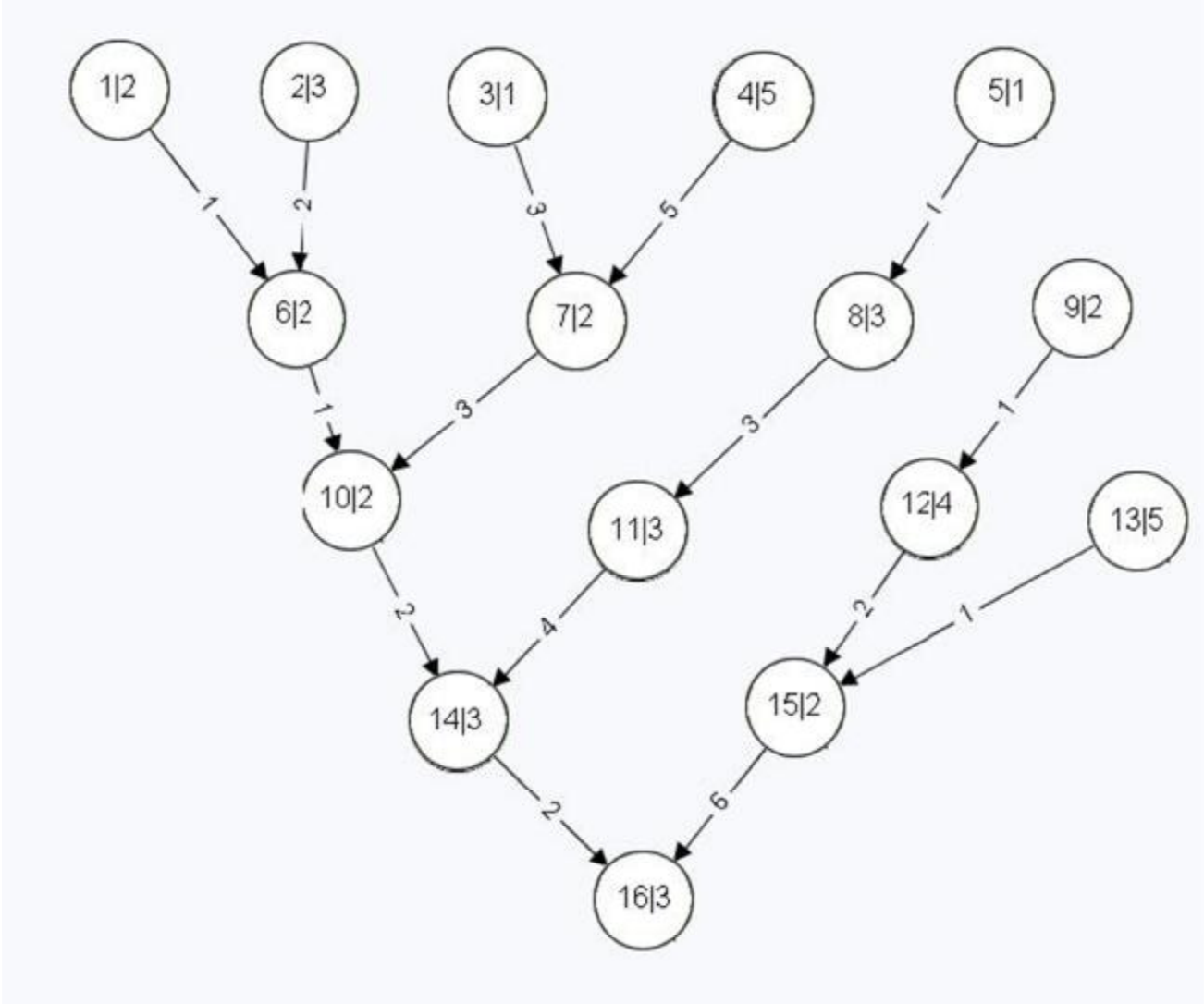


Рис. 5

Внесем наш граф в разработанное приложение для моделирования рис 6.

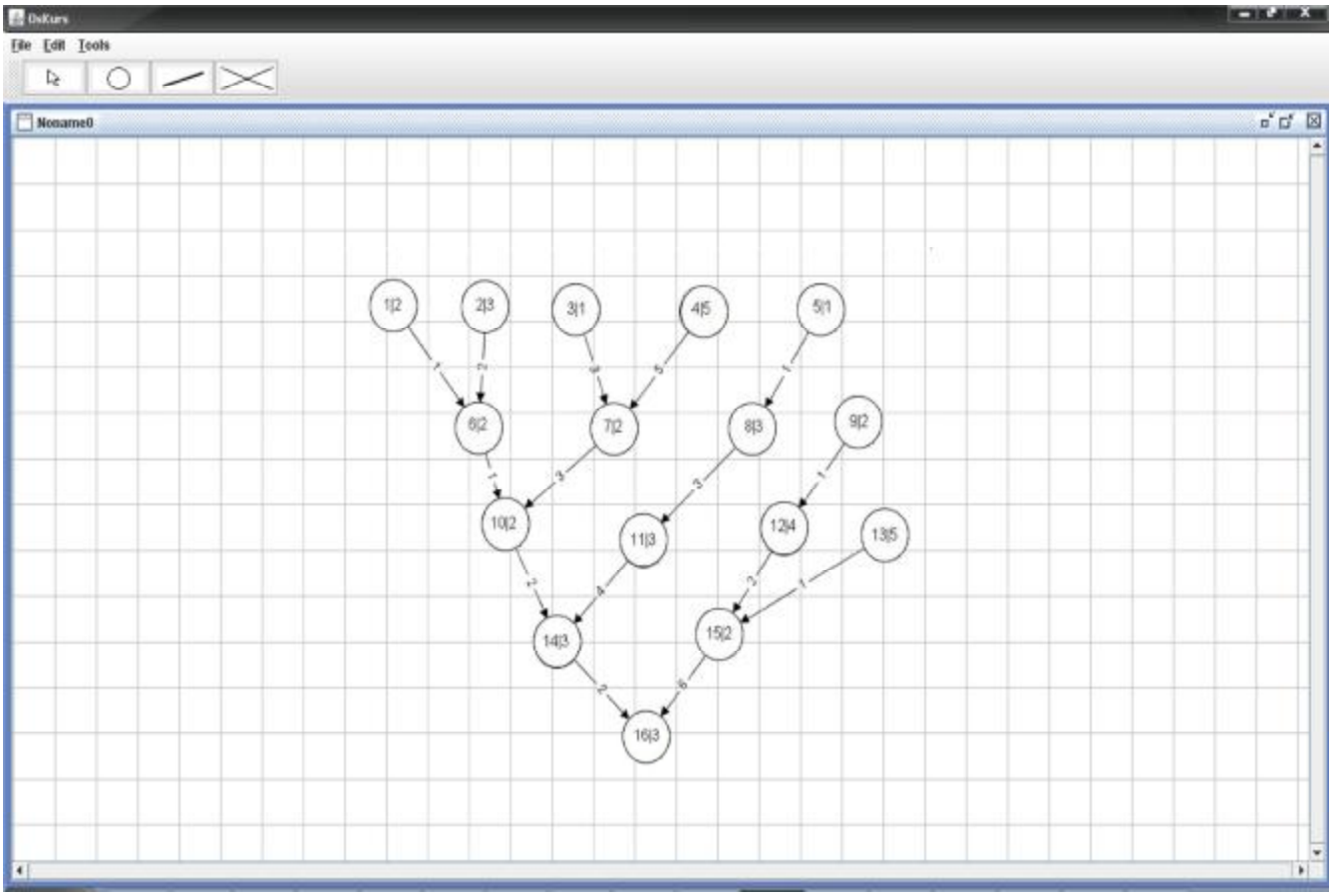


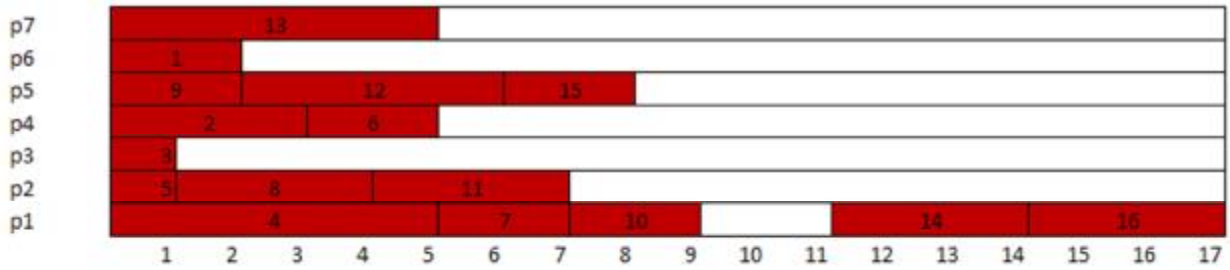
Рис 6.

Для различных конфигураций системы, были получены различные результаты

Для первого набора (рис. 7) результат моделирования указан на рис.8

p1	p2	p3	p4	p5	p6	p7
1	1	1	1	1	1	1

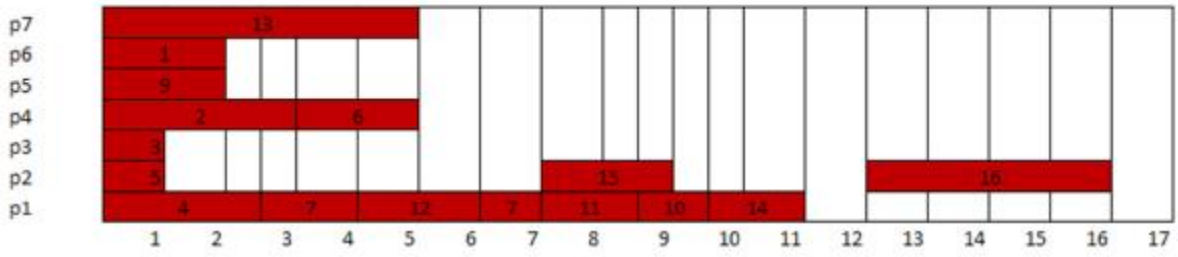
Рис. 7



Для второго набора (рис.9) результат указан на рис.10

p1	p2	p3	p4	p5	p6	p7
2	1	1	1	1	1	1

Рис .9



Выводы

В данной курсовой работе был разработан алгоритм динамического распределения задач на ресурсы вычислительной системы с топологией полносвязной неоднородной структуры. Для иллюстрации работы разработанного алгоритма была построена моделирующая программа, которая показывает процесс загрузки пользовательского графа задачи на систему. При выполнении работы был использован алгоритм планирования (HEFT).

5. Список использованной литературы

1. Haluk Topcuoglu, Salim Hariri, Min-You Wu "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", IEEE Trans. Parallel and Distributed Systems vol. 13, no. 3, Mar. 2002, pp. 260-274

M.R. Gary and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., 1979.

2. Конспект лекций по курсу «Операционные системы»
3. Таненбаум Э. Современные операционные системы. 2-е изд. – СПб.: Питер, 2002. – 1040 с.
4. Кудин А.В., Линев А.А. Архитектура и операционные системы параллельных вычислительных систем. Учебно-методические материалы – Нижний Новгород, 2007

Приложения

Приложение А.

Исходный код

```
package editor;


import com.sun.corba.se.impl.interceptors.InterceptorInvoker;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Toolkit;
import java.awt.Dimension;
import java.awt.event.FocusListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.beans.PropertyVetoException;
import javax.swing.JComponent;
import javax.swing.JDesktopPane;

import javax.swing.JFrame;
import javax.swing.JInternalFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JToolBar;
import javax.swing.event.InternalFrameAdapter;
import javax.swing.event.InternalFrameEvent;
import javax.swing.event.InternalFrameListener;



public class MainFrame extends JFrame{
```

```
private JMenuBar menuBar;
private JToolBar toolBar;
private JDesktopPane desktop;

public final static int stdIntFrameWidth = 700;
public final static int stdIntFrameHeight = 500;

public MainFrame() {
    super("OsKurs");

    //Set Size and Position of Frame
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    int width = (int)Math.round(0.75*toolkit.getScreenSize().width);
    int height = (int)Math.round(0.75*toolkit.getScreenSize().height);
    Dimension frameSize = new Dimension(width,height);
    this.setSize(frameSize);
    int x = Math.round((toolkit.getScreenSize().width - width)/2);
    int y = Math.round((toolkit.getScreenSize().height - height)/2);
    this.setLocation(x,y);

    this.setLayout(new BorderLayout(2,3));

    //MENU BAR
    menuBar = new JMenuBar();
    this.fillMenu();
    this.setJMenuBar(menuBar);

    //TOOL BAR
    toolBar = new JToolBar();
    this.fillToolBar();
```

```
        this.add(toolBar,BorderLayout.NORTH);



        //Desktop
        desktop = new JDesktopPane();
        desktop.setBackground(Color.GRAY);
        this.add(desktop);

        this.addWindowListener(new WindowsExit());
        this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        this.setVisible(true);
        this.addFirstInternalFrame();
        Mediator.getInstance().regMainFrame(this);
    }

    private void addFirstInternalFrame(){
        JInternalFrame iFrame = new JInternalFrame("Noname0",true,true,true,true);
        iFrame.reshape(0,0,desktop.getSize().width,desktop.getSize().height);
        iFrame.add(new JScrollPane(new
BlockPaintPanel(Mediator.elementWidth,Mediator.elementHeight),JScrollPane.VERTICAL_SC
ROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS));
        iFrame.addInternalFrameListener(new InternalFrameExit());
        iFrame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        iFrame.setVisible(true);
        this.desktop.add(iFrame);
        desktop.setSelectedFrame(iFrame);

    }

    public void addInternalFrame(JInternalFrame intFrame,JComponent comp){
```

```
        intFrame.add(new
JScrollPane(comp,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONT
AL_SCROLLBAR_ALWAYS));
        intFrame.reshape(0,0,desktop.getSize().width,desktop.getSize().height);
        intFrame.addInternalFrameListener(new InternalFrameExit());
        intFrame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        intFrame.setVisible(true);
        this.desktop.add(intFrame);
        try {
            intFrame.setSelected(true);
        }catch(PropertyVetoException ex){}

    }

    public JInternalFrame[] getAllInternalFrames(){
        return this.desktop.getAllFrames();
    }

    public JInternalFrame getSelectedFrame(){
        return this.desktop.getSelectedFrame();
    }

    private void fillMenu(){
        ActionBuilder builder = ActionBuilder.getInstance();

        //Menu "File"
        JMenu file = new JMenu("File");
        file.setMnemonic('F');
        file.add(builder.getAction("New","Create new file",'N',"icons//new.gif"));
        file.addSeparator();
        file.add(builder.getAction("Open","Open existing file",'O',"icons//open.gif"));
```

```
  file.addSeparator();
  file.add(builder.getAction("SaveAs","Save to another file",'A',"icons//save_as.gif"));
  file.addSeparator();
  file.add(builder.getAction("Exit","Exit from application",'E'));
  this.menuBar.add(file);

//Menu "Edit"
  JMenu edit = new JMenu("Edit");
  edit.add(builder.getAction("Delete","Delete selected elements",'d',"icons//delete.gif"));
  edit.setMnemonic('E');
  this.menuBar.add(edit);


  //Menu "Tools "
  JMenu tool = new JMenu("Tools");
  tool.setMnemonic('T');
  //Menu "Tools-->Automate"
  tool.add(builder.getAction("Mark Mili","Mark automate Mili",'M'));
  tool.add(builder.getAction("Unmark Mili","Unmark automate Mili",'M'));
  tool.addSeparator();
  tool.add(builder.getAction("Build Mili","Build automate Mili"));
  tool.add(builder.getAction("Build secure Mili","Build automate Mili"));
  tool.add(builder.getAction("Save Mili","Save automate Mili"));
  tool.add(builder.getAction("Open Mili","Open automate Mili"));
  tool.addSeparator();
  tool.add(builder.getAction("Build transmition table","Build transmition table"));
  tool.add(builder.getAction("Save transmition table","Save transmition table"));
  tool.add(builder.getAction("Open transmition table","Open transmition table"));
  tool.addSeparator();
  tool.add(builder.getAction("Build vhdl file","Build vhdl"));
```

```
        this.menuBar.add(tool);
    }

    private void fillToolBar(){
        ActionBuilder builder = ActionBuilder.getInstance();
        this.toolBar.add(builder.getAction("Cancel all","Cancel all
actions","icons//cancel_all.gif"));
        this.toolBar.add(builder.getAction("Operation Block","Create operation
block","icons//operation.gif"));
        this.toolBar.add(builder.getAction("Condition Block","Create operation
block","icons//condition.gif"));
//        this.toolBar.add(builder.getAction("Beg_end Block","Create begin
block","icons//beg_end.gif"));
        this.toolBar.add(builder.getAction("Wire","Create operation block","icons//wire.gif"));
    }

    private class WindowsExit extends WindowAdapter{

        public void windowClosing(WindowEvent e) {
            int res = JOptionPane.showConfirmDialog(MainFrame.this
                        ,"Are you realy want exit","Confirm",JOptionPane.YES_NO_OPTION);
            if(res == 0) System.exit(0);
        }
    }

    private class InternalFrameExit extends InternalFrameAdapter{

        public void internalFrameClosing(InternalFrameEvent e) {
            int res = JOptionPane.showConfirmDialog(MainFrame.this
                        ,"Are you realy want exit","Confirm",JOptionPane.YES_NO_OPTION);
            if(res == 0) e.getInternalFrame().dispose();
        }
```

```
    }
}
package editor;

import analizator.*;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Color;
import java.awt.event.MouseMotionAdapter;
import java.awt.Point;
import java.io.Serializable;

import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;




public class BlockPaintPanel extends JPanel implements Serializable{

    public static final int NORMAL = 0;
```

```
public static final int OPERATION = 1;
public static final int CONDITION = 2;
public static final int WIRE = 3;
public static final int BEG_END = 4;

private int mouseStatus;

private ArrayList<AbstractPaintElement> elementContent;
private ArrayList<BlockLine> lineContent;
private BlockLine lastLine;

private AbstractPaintElement selectedElement;
private BlockLine selectedLine;

private AbstractPaintElement bufferedElement;

private JPopupMenu popupMenu;

private BlockPaintPanelModel model;

private int elementCount;
private int automateCount;

private int elementWidth;
private int elementHeight;

public BlockPaintPanel(int elementWidth,int elementHeight){
    this.elementWidth = elementWidth;
    this.elementHeight = elementHeight;
    this.selectedElement = null;
    this.elementContent = new ArrayList<AbstractPaintElement>();
```

```
    this.lineContent = new ArrayList<BlockLine>();
    this.lastLine = null;
    this.popupMenu = new JPopupMenu();
    this.model = new BlockPaintPanelModel();
    this.elementCount = 0;
    this.setBackground(Color.white);
    this.addKeyListener(new KeyElementListener());
    this.addMouseListener(new MousePaintListener());
    this.addMouseMotionListener(new MouseElementMotionListener());
    this.createPopupMenu();
    this.menuRefrech();
}

public void setSize(Dimension dim){
    if(this.getSize().height < dim.height)
        super.setSize(new Dimension(getSize().width,dim.height));
    if(this.getSize().width < dim.width)
        super.setSize(new Dimension(dim.width,getSize().height));
}


public void paintComponent(Graphics g){
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    //grid
    g2.setColor(Color.LIGHT_GRAY);
    int delta = 40;
    int x = 1,y = 1;
    int width = this.getSize().width;
    int height = this.getSize().height;
    while(x < width){
```

```
        g2.drawLine(x,0,x,height);
        x += delta;
    }
    while(y < height){
      g2.drawLine(0,y,width,y);
      y += delta;
    }
    g.setColor(Color.BLACK);
    //lines
    for(BlockLine pl : lineContent)
        pl.paint(g);
    if(lastLine!= null)lastLine.paint(g);
    //elements
    for(AbstractPaintElement ape : elementContent)
        ape.paint(g);
}

public void setMouseStatus(int mouseStatus){
    Toolkit tool = Toolkit.getDefaultToolkit();
    Image img = null;
    switch(mouseStatus){
        case BlockPaintPanel.OPERATION:
            img = tool.getImage("icons//cursor_rect.gif");
            break;
        case BlockPaintPanel.CONDITION:
            img = tool.getImage("icons//cursor_romb.gif");
            break;
        case BlockPaintPanel.BEG_END:
            img = tool.getImage("icons//cursor_ellipse.gif");
            break;
        case BlockPaintPanel.WIRE:
```

```
            img = tool.getImage("icons//cursor_wire.gif");
            break;
        case BlockPaintPanel.NORMAL:
            this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
            break;
    }
    if(img != null){
        Cursor cursor = tool.createCustomCursor(img,new Point(0,0),"Cursor_Common");
        this.setCursor(cursor);
    }
    this.mouseStatus = mouseStatus;
    if(mouseStatus != WIRE) this.lastLine = null;
}

public int getMouseStatus(){
    return this.mouseStatus;
}

public void delete(){
    if(selectedLine != null) deleteLine(selectedLine);
    if(selectedElement != null) deleteElement(selectedElement);
    repaint();
}

public void copy(){
    if(selectedElement != null) bufferedElement = selectedElement;
}

public void cut(){
    if(selectedElement != null){
        bufferedElement = selectedElement;
```

```
        deleteElement(selectedElement);
    }
    repaint();
}

public void paste(Point point){
    if(bufferedElement != null){
        AbstractPaintElement next = (AbstractPaintElement) bufferedElement.clone();
        next.setAnchor(point);
        this.elementContent.add(next);
        model.add(next);
    }
    repaint();
}

public void showPropDialog(){
    if(selectedElement != null) selectedElement.showPropertiesDialog();
    repaint();
}

public void setModel(BlockPaintPanelModel model){
    this.model = model;
    ArrayList<AbstractBlock> blocks = this.model.getAll();
    this.elementCount = blocks.size();
    AbstractPaintElement ape = null;
    AbstractBlock prev = null;
    Point point = new Point (this.getSize().width/2 - elementWidth/2,10);
    for(AbstractBlock bl : blocks){
        //find yangest parent
        for(AbstractBlock bl1 : model.findParentBlock(bl.getID()))
            if(bl1.getID() < bl.getID()) prev = bl1;
```

```
        //BEGIN
        if(bl instanceof BeginBlock){
            ape = new BegEndPaintElement(this.elementWidth,this.elementHeight,point,"Begin");
            ape.setID(bl.getID());

        //END
        }else  if(bl instanceof EndBlock){
            for(AbstractPaintElement ape1 : this.elementContent)
                if(ape1.getAnchor().y > ape.getAnchor().y) ape = ape1;
            point = new Point(this.findElement(0).getAnchor().x , ape.getAnchor().y +
2*this.elementHeight);
            ape = new BegEndPaintElement(this.elementWidth,this.elementHeight,point,"End");
            ape.setID(bl.getID());

        //OPERATION
        }else  if(bl instanceof OperationBlock){
            ape = this.findElement(prev.getID());
            if(prev instanceof ConditionBlock && prev.getNextFalse().getID() == bl.getID())
                point = new Point(ape.getAnchor().x +
(int)(1.2*this.elementWidth),ape.getAnchor().y + 2*this.elementHeight);
            else
                point = new Point(ape.getAnchor().x,ape.getAnchor().y + 2*this.elementHeight);
            ape = new
OperationPaintElement(this.elementWidth,this.elementHeight,point,bl.getOperation());
            ape.setID(bl.getID());

        //CONDITION
        }else  if(bl instanceof ConditionBlock){
            ape = this.findElement(prev.getID());
            if(prev instanceof ConditionBlock && prev.getNextFalse().getID() == bl.getID())
                point = new Point(ape.getAnchor().x +
(int)(1.5*this.elementWidth),ape.getAnchor().y + 2*this.elementHeight);
```

```
            else{
                point = new Point(ape.getAnchor().x,ape.getAnchor().y + 2*this.elementHeight);
                for(AbstractPaintElement ape1 : this.elementContent)
                    if(ape1.getAnchor().y > ape.getAnchor().y) ape = ape1;
                point.y = ape.getAnchor().y + 2*this.elementHeight;
            }
            ape = new
ConditionPaintElement(this.elementWidth,this.elementHeight,point,bl.getOperation());
            ape.setID(bl.getID());
          }
          this.elementContent.add(ape);
       }
       this.setSize(this.getSize().width,findElement(model.getEnd().getID()).getAnchor().y +
elementHeight);
       this.lineContent = this.createLines();
       repaint();
    }

    public BlockPaintPanelModel getModel(){
       return model;
    }

   public boolean markAutomateMura(){
      this.automateCount = 0;
      if(!this.model.areYouOk()) return false;
      this.markElement(this.model.getRoot());
      this.findElement(this.model.getEnd().getID()).setAutomateLable("Z" + 0);
      this.repaint();
      return true;
   }
```

```
  private void markElement(AbstractBlock block){
     if(block instanceof EndBlock) return;
     if(findElement(block.getID()).getAutomateLable()!= null) return;
     if(!(block instanceof ConditionBlock)){
        this.findElement(block.getID()).setAutomateLable("Z" + this.automateCount++);
     }
     this.markElement(block.getNextTrue());
     if(block.getNextFalse() != null) this.markElement(block.getNextFalse());
  }

  public void unmarkAutomateMura(){
     for(AbstractPaintElement ape : elementContent)
        ape.setAutomateLable(null);
     repaint();
  }

  public boolean markAutomateMili(){
     this.automateCount = 0;
     if(!this.model.areYouOk()) return false;
     this.markLine(this.model.getRoot());

this.findOutputLine(model.findParentBlock(model.getEnd().getID()).get(0).getID()).setAutomat
eLable("A" + 0);
     this.repaint();
     return true;
  }

  private void markLine(AbstractBlock block){
     ArrayList<AbstractBlock> parents = this.model.findParentBlock(block.getID());
     if(block instanceof EndBlock) return;
     if(parents != null)
```

```
        for(AbstractBlock bl : parents)
            if(!(bl instanceof ConditionBlock)){
                if(this.findOutputLine(bl.getID()).getAutomateLable() != null) return;
                this.findOutputLine(bl.getID()).setAutomateLable("A" + this.automateCount);
                this.automateCount ++;
                break;
            }
    if(block.getNextTrue()!=null) this.markLine(block.getNextTrue());
    if(block.getNextFalse() != null) this.markLine(block.getNextFalse());
}

public void unmarkAutomateMili(){
    for(BlockLine line : lineContent)
        line.setAutomateLable(null);
    repaint();
}

public void unmark(){
    this.unmarkAutomateMili();
    this.unmarkAutomateMura();
    repaint();
}



private ArrayList<BlockLine> createLines(){
     ArrayList<BlockLine> pl = new ArrayList<BlockLine>();
     for(AbstractBlock block: model.getAll()){
         if(block.getNextTrue() != null)
             pl.add(new
BlockLine(this.findElement(block.getID()),this.findElement(block.getNextTrue().getID()),true));
         if(block.getNextFalse() != null)
```

```
            pl.add(new
BlockLine(this.findElement(block.getID()),this.findElement(block.getNextFalse().getID()),false)
);
        }
        return pl;
    }

    private void deleteLine(BlockLine pl){
        model.setNext(pl.getFirst().getID(),-1,pl.getDirection());
        this.lineContent.remove(pl);
        if(selectedLine == pl) setSelectedLine(null);
    }

    private void deleteElement(AbstractPaintElement ape){
        ArrayList<BlockLine> line = getLines(ape);
        for(BlockLine pl:line)
            deleteLine(pl);
        this.elementContent.remove(ape);
        model.delete(ape.getID());
        if(selectedElement == ape) setSelectedElement(null);
    }


    private void setSelectedElement(AbstractPaintElement ape){
        if(this.selectedElement != null) selectedElement.setColor(Color.BLACK);
        this.selectedElement = ape;
        if(selectedElement != null) selectedElement.setColor(Color.RED);
        menuRefrech();
    }

    private void setSelectedLine(BlockLine pl){
```

```
    if(this.selectedLine != null) selectedLine.setColor(Color.BLACK);
    this.selectedLine = pl;
    if(selectedLine != null) selectedLine.setColor(Color.RED);
    menuRefrech();
}

private BlockLine getLine(Point p){
   for(BlockLine pl :lineContent)
      if(pl.isItYour(p)) return pl;
   return null;
}

private ArrayList<BlockLine> getLines(AbstractPaintElement ape){
    ArrayList<BlockLine> result = new ArrayList<BlockLine>();
    for(BlockLine pl: this.lineContent)
       if(pl.getFirst() == ape || pl.getSecond() == ape) result.add(pl);
    return result;
}

private AbstractPaintElement getElement(Point point){
    AbstractPaintElement result = null;
    int deltaX = 0;
    int deltaY = 0;
    for(AbstractPaintElement ape : elementContent)
      if(ape.isItYour(point)){
          if(result != null && (point.x - result.getAnchor().x ) > (point.x - ape.getAnchor().x))
             result = ape;
          if(result == null) result = ape;

      }
    return result;
```

```
}

private void menuRefrech(){
    ActionBuilder builder = ActionBuilder.getInstance();

    if(selectedLine != null || selectedElement != null)
        builder.getAction("Delete","Delete selected elements").setEnabled(true);
    else
        builder.getAction("Delete","Delete selected elements").setEnabled(false);

    if(bufferedElement != null)
        builder.getAction("Paste","Delete selected elements").setEnabled(true);
    else
        builder.getAction("Paste","Delete selected elements").setEnabled(false);

    if(selectedElement != null){
        builder.getAction("Cut","Cut selected elements").setEnabled(true);
        builder.getAction("Copy","Copy selected elements").setEnabled(true);
        builder.getAction("Properties","Show properties dialog").setEnabled(true);
    }else{
        builder.getAction("Cut","Cut selected elements").setEnabled(false);
        builder.getAction("Copy","Copy selected elements").setEnabled(false);
        builder.getAction("Properties","Show properties dialog").setEnabled(false);
    }
}

private void createPopupMenu(){
    ActionBuilder builder = ActionBuilder.getInstance();
    popupMenu.add(builder.getAction("Cut","Cut selected elements","icons//cut.gif"));
    popupMenu.add(builder.getAction("Copy","Copy selected elements","icons//copy.gif"));
    popupMenu.add(builder.getAction("Paste","Paste selected elements","icons//paste.gif"));
```

```
        popupMenu.add(builder.getAction("Delete","Delete selected
elements","icons//delete.gif"));
        popupMenu.addSeparator();
        popupMenu.add(builder.getAction("Properties","Show properties dialog",'P'));
    }

    private AbstractPaintElement findElement(int id){
        for(AbstractPaintElement ape:this.elementContent)
            if(ape.getID() == id ) return ape;
        return null;
    }

    private BlockLine findOutputLine(int id){
        for(BlockLine line:this.lineContent)
            if(line.getFirst().getID() ==  id) return line;
        return null;
    }

    private class MousePaintListener extends MouseAdapter{

        public void mouseClicked(MouseEvent source) {

            //right button
            if(source.getButton() == 3 ){
                lastLine = null;
                setMouseStatus(BlockPaintPanel.NORMAL);
                setSelectedElement(getElement(source.getPoint()));
                setSelectedLine(getLine(source.getPoint()));

ActionBuilder.getInstance().getAction("Paste","g").putValue("Point",source.getPoint());
```

```
BlockPaintPanel.this.popupMenu.show(BlockPaintPanel.this,source.getPoint().x,source.getPoint
().y);
        }else{
        //LEFT BUTTON
           AbstractPaintElement element = null;
           setSelectedElement(null);
           setSelectedLine(null);
           Point anchor = new Point(source.getPoint().x-elementWidth/2,source.getPoint().y-
elementHeight/2);
           switch(mouseStatus){
             case BlockPaintPanel.OPERATION:
                element = new OperationPaintElement(elementWidth,elementHeight,anchor);
                element.setID(elementCount++);
                element.paint(getGraphics());
                if(element.showPropertiesDialog()){
                   elementContent.add(element);
                   model.add(element);
                }
                break;
             case BlockPaintPanel.CONDITION:
                element = new ConditionPaintElement(elementWidth,elementHeight,anchor);
                element.setID(elementCount++);
                element.paint(getGraphics());
                if(element.showPropertiesDialog()){
                   elementContent.add(element);
                   model.add(element);
                }
                break;

              case BlockPaintPanel.BEG_END:
                element = new BegEndPaintElement(elementWidth,elementHeight,anchor);
```

```
                element.setID(elementCount++);
                element.paint(getGraphics());
                if(element.showPropertiesDialog()){
                   if(element.getLable().equals("End") && model.getEnd() == null){
                      elementContent.add(element);
                      model.add(element);
                      model.setEnd(element.getID());
                   }else if(element.getLable().equals("Begin") && model.getRoot() == null){
                      elementContent.add(element);
                      model.add(element);
                      model.setRoot(element.getID());
                   }else
                       JOptionPane.showMessageDialog(Mediator.getInstance().getMainFrame()
                              ,"You already have such type of
block","Error",JOptionPane.ERROR_MESSAGE);
                }
                break;

             case BlockPaintPanel.WIRE:
                AbstractPaintElement ape = getElement(source.getPoint());
                setSelectedElement(ape);
                //New Line
                if(lastLine == null){
                   if(ape == null) return;
                   AbstractBlock block = model.get(ape.getID());
                   if(ape.getLable().equals("End")){
                      JOptionPane.showMessageDialog(Mediator.getInstance().getMainFrame()
                              ,"This is End block. You can not do
it","Error",JOptionPane.ERROR_MESSAGE);
                      return;
                   }
```

```
                else if(block.areYouOk()){
                    JOptionPane.showMessageDialog(Mediator.getInstance().getMainFrame()
                            ,"This block already have
connection","Error",JOptionPane.ERROR_MESSAGE);
                    return;
                }
                if(ape instanceof ConditionPaintElement){
                    boolean direction = true;
                    if(block.getNextTrue() == null  && block.getNextFalse() == null){
                        String str = "Choose \"Yes\" for true direction\n Choose \"No\" for false
direction";
                        int res =
JOptionPane.showConfirmDialog(Mediator.getInstance().getMainFrame(),str,"Direction",JOptio
nPane.YES_NO_OPTION,JOptionPane.QUESTION_MESSAGE);
                        if(res == 0) direction = true;
                        else direction = false;
                    }else if(block.getNextTrue() == null) direction = true;
                     else if(block.getNextFalse() == null) direction  = false;
                    lastLine = new BlockLine(ape,direction);
                }
                else lastLine = new BlockLine(ape,true);
            }
            //Continue line creation
            else
                if(ape != null){
                    if(ape.getLable().equals("Begin")){
                        JOptionPane.showMessageDialog(Mediator.getInstance().getMainFrame()
                                ,"This is begin block. You can not do
it","Error",JOptionPane.ERROR_MESSAGE);
                        return;
                    }
                    if(ape == lastLine.getFirst()){
                        JOptionPane.showMessageDialog(Mediator.getInstance().getMainFrame()
```

```
                                ,"You can not do it","Error",JOptionPane.ERROR_MESSAGE);
                            return;
                        }
                        lastLine.setLastElement(ape);
                        if(lastLine.getDirection())
model.get(lastLine.getFirst().getID()).setNextTrue(model.get(ape.getID()));
                        else
model.get(lastLine.getFirst().getID()).setNextFalse(model.get(ape.getID()));
                        lineContent.add(lastLine);
                        lastLine = null;
                    }
                break;
            case BlockPaintPanel.NORMAL:
                setSelectedElement(getElement(source.getPoint()));
                setSelectedLine(getLine(source.getPoint()));
                break;
          }
        }
        repaint();

    }//mouseClicked
}//MousePaintListene

private class MouseElementMotionListener extends MouseMotionAdapter{

    public void mouseMoved(MouseEvent source){

    }//mouseMoved

    public void mouseDragged(MouseEvent source){
        AbstractPaintElement ape = getElement(source.getPoint());
```

```
            if(ape == null) return;
            if(lastLine != null) return;
            setMouseStatus(BlockPaintPanel.NORMAL);
            setSelectedElement(ape);
            Point next = source.getPoint();
            if(next.x + elementWidth > getSize().width) setSize(getSize().width +
5,getSize().height);
            if(next.y + elementHeight > getSize().height) setSize(getSize().width,getSize().height +
5);
            ArrayList<BlockLine> pl = getLines(ape);
            ape.setAnchor(new Point(next.x-ape.getWidth()/2,next.y-ape.getHeight()/2));
            for(BlockLine paint : pl)
               paint.refresh();
            repaint();
        }//mouseDragged

    }//MouseElementMotionListener

    private class KeyElementListener extends KeyAdapter{

        public void keyPressed(KeyEvent source){



        }//keyPressed
    }//KeyElementListener
}//PainPanel
```