Национальный технический университет Украины

«Киевский политехнический институт»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

**Лабораторная работа №3**

*по курсу «Операционные системы»*

**Исследование дисциплин обслуживания заявок при ограниченных ресурсах**

Выполнил

студент группы ИВ-73

Захожий Игорь

Номер зачетной книжки: 7308

Киев-2010

**Задание на работу**

1. Згідно варіанту виконати по тактову візуалізацію системи обслуговування заявок з розподіленим ресурсом, параметрами системи є діапазон ваг заявок, діапазон інтервалів виникнення наступної заявки (інтенсивність вхідного потоку), діапазон пріоритетів (для пріоритетних дисциплін обслуговування). В систему може надходити будь-яка кількість заявок.
2. Якщо система обслуговування з чергами, то кількість черг не більша 32. Якщо система обслуговування з пріоритетами, то число пріоритетів не більш ніж 32.
3. Побудувати графіки залежності середнього часу очікування від інтенсивності вхідного потоку заявок та залежність проценту простою ресурсу від інтенсивності вхідного потоку заявок.
4. Для пріоритетних систем побудувати графік залежності середнього часу очікування від пріоритету при фіксованої інтенсивності вхідного потоку заявок. Для систем без пріоритетів побудувати графік дисперсії часу очікування при фіксованій інтенсивності вхідного потоку заявок.
5. Пояснити форму графіків.

**Определение варианта**

8 mod 22 + 1 = 9 => Вариант №9 – Алгоритм Корбато.

**Описание работы дисциплины обслуживания**

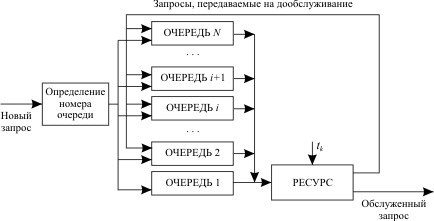
Выделяемый программе квант времени на обслуживание возрастает с увеличением номера очереди обычно по правилу:

tki = 2i-1 · tk

где tk - квант времени, выделяемый для программ из очереди 1.

Считается, что продолжительность выполнения программы приблизительно пропорциональна ее длине. По крайней мере, от длины программы прямо зависит время, затрачиваемое на передачу программы между ОЗУ и внешним ЗУ при ее активизации.

Определение номера очереди, в которую поступает программа при первоначальной загрузке, осуществляется по *алгоритму планирования Корбато*: программа сразу поступает в очередь i = [log2 lp/ltk + 1], где lp - длина программы в байтах; ltk - число байт, которые могут быть переданы между ОЗУ и внешней памятью за время tk ([рис. 1](http://www.intuit.ru/department/hardware/archhard2/13/#image.13.5)).

  
*Рис. 1.*  Схема распределения ресурса при многоочередной дисциплине обслуживания со статическим указанием приоритетов программ

Эта *дисциплина* позволяет сократить количество системных переключений за счет того, что программам, требующим большего времени решения, будут предоставляться достаточно большие кванты времени уже при первом занятии ими ресурса (нерационально программе, которая требует для своего решения 1 час времени, первоначально выделять квант в 1 мс).

**Листинг программы**

**package** os.lab3;

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

**public** **class** ProgramStart {

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

**if** (System.*getProperty*("os.name").contains("Windows")) {

**try** {

UIManager.*setLookAndFeel*("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

} **catch** (ClassNotFoundException e) {

} **catch** (InstantiationException e) {

} **catch** (IllegalAccessException e) {

} **catch** (UnsupportedLookAndFeelException e) {}

}

MainFrame frame = **new** MainFrame();

frame.setVisible(**true**);

}

}

**package** os.lab3;

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

**import** java.awt.\*;

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

**public** **class** MainFrame **extends** JFrame {

**private** JTabbedPane tabbedPane;

**public** MainFrame() {

**super**();

setTitle("Лабораторная работа №3 - Алгоритм Корбато");

setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

setMinimumSize(**new** Dimension(800, 600));

setExtendedState(JFrame.*MAXIMIZED\_BOTH*);

tabbedPane = **new** JTabbedPane();

JPanel vPanel = **new** VisualizationPanel();

WaitTimePanel g1Panel = **new** WaitTimePanel();

CPUDowntimePanel g2Panel = **new** CPUDowntimePanel();

PriorityPanel g3Panel = **new** PriorityPanel();

tabbedPane.add("Потактовая визуализация", vPanel);

tabbedPane.add("График среднего времени ожидания от интенсивности", g1Panel);

tabbedPane.add("График процента простоя ресурса от интенсивности", g2Panel);

tabbedPane.add("График среднего времени ожидания от приоритета", g3Panel);

tabbedPane.setSelectedIndex(0);

add(tabbedPane);

}

**private** **class** VisualizationPanel **extends** JPanel {

**private** JTextField lambdaField;

**private** JTextField minLpField;

**private** JTextField maxLpField;

**private** JTextField tkField;

**private** JTextField ltkField;

**private** JTextField queuesCountField;

**private** JTextField stepsField;

**private** JTextArea consoleArea;

**private** JButton clearButton;

**private** Corbato corbato;

**private** **int** step;

**public** VisualizationPanel() {

**super**();

setLayout(**new** BorderLayout());

JPanel parametersPanel = **new** JPanel();

parametersPanel.setLayout(**new** GridLayout(8, 1));

JPanel tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

tempPanel.add(**new** JLabel("Интенсивность входного потока заявок"));

lambdaField = **new** JTextField();

lambdaField.setColumns(10);

tempPanel.add(lambdaField);

parametersPanel.add(tempPanel);

tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

tempPanel.add(**new** JLabel("Минимальная длина программы"));

minLpField = **new** JTextField();

minLpField.setColumns(10);

tempPanel.add(minLpField);

parametersPanel.add(tempPanel);

tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

tempPanel.add(**new** JLabel("Максимальная длина программы"));

maxLpField = **new** JTextField();

maxLpField.setColumns(10);

tempPanel.add(maxLpField);

parametersPanel.add(tempPanel);

tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

tempPanel.add(**new** JLabel("Время Tk"));

tkField = **new** JTextField();

tkField.setColumns(10);

tempPanel.add(tkField);

parametersPanel.add(tempPanel);

tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

tempPanel.add(**new** JLabel("Количество байт, переданных за время Tk"));

ltkField = **new** JTextField();

ltkField.setColumns(10);

tempPanel.add(ltkField);

parametersPanel.add(tempPanel);

tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

tempPanel.add(**new** JLabel("Количество очередей"));

queuesCountField = **new** JTextField();

queuesCountField.setColumns(10);

tempPanel.add(queuesCountField);

parametersPanel.add(tempPanel);

tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

tempPanel.add(**new** JLabel("Количество тактов при моделировании"));

stepsField = **new** JTextField();

stepsField.setColumns(10);

tempPanel.add(stepsField);

parametersPanel.add(tempPanel);

tempPanel = **new** JPanel();

tempPanel.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

clearButton = **new** JButton(**new** AbstractAction() {

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

lambdaField.setEnabled(**true**);

minLpField.setEnabled(**true**);

maxLpField.setEnabled(**true**);

tkField.setEnabled(**true**);

ltkField.setEnabled(**true**);

queuesCountField.setEnabled(**true**);

consoleArea.setText("");

((JButton)event.getSource()).setEnabled(**false**);

corbato = **null**;

}

});

clearButton.setText("Остановить");

clearButton.setEnabled(**false**);

tempPanel.add(clearButton);

JButton stepButton = **new** JButton(**new** AbstractAction() {

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

**if** (corbato == **null**) {

lambdaField.setEnabled(**false**);

minLpField.setEnabled(**false**);

maxLpField.setEnabled(**false**);

tkField.setEnabled(**false**);

ltkField.setEnabled(**false**);

queuesCountField.setEnabled(**false**);

clearButton.setEnabled(**true**);

step = 0;

corbato = **new** Corbato(

Integer.*valueOf*(queuesCountField.getText()),

Integer.*valueOf*(tkField.getText()),

Integer.*valueOf*(ltkField.getText()),

Double.*valueOf*(lambdaField.getText()),

Integer.*valueOf*(minLpField.getText()),

Integer.*valueOf*(maxLpField.getText()));

}

consoleArea.setText(consoleArea.getText() + "Такт " +

String.*valueOf*(step + 1) + ":\n" + corbato.tactWithLog() + "\n");

step++;

}

});

stepButton.setText("Такт");

tempPanel.add(stepButton);

JButton modellingButton = **new** JButton(**new** AbstractAction() {

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

**if** (corbato == **null**) {

lambdaField.setEnabled(**false**);

minLpField.setEnabled(**false**);

maxLpField.setEnabled(**false**);

tkField.setEnabled(**false**);

ltkField.setEnabled(**false**);

queuesCountField.setEnabled(**false**);

clearButton.setEnabled(**true**);

step = 0;

corbato = **new** Corbato(

Integer.*valueOf*(queuesCountField.getText()),

Integer.*valueOf*(tkField.getText()),

Integer.*valueOf*(ltkField.getText()),

Double.*valueOf*(lambdaField.getText()),

Integer.*valueOf*(minLpField.getText()),

Integer.*valueOf*(maxLpField.getText()));

}

**int** prevStep = step;

**for** (step = step; step < (prevStep + Integer.*valueOf*(stepsField.getText())); step++) {

consoleArea.setText(consoleArea.getText() + "Такт " +

String.*valueOf*(step + 1) + ":\n" + corbato.tactWithLog() + "\n");

}

}

});

modellingButton.setText("Моделирование");

tempPanel.add(modellingButton);

parametersPanel.add(tempPanel);

add(parametersPanel, BorderLayout.*WEST*);

consoleArea = **new** JTextArea();

consoleArea.setBackground(Color.*DARK\_GRAY*);

Font consoleFont = **new** Font("Monospaced", Font.*PLAIN*, 14);

consoleArea.setEditable(**false**);

consoleArea.setAutoscrolls(**true**);

consoleArea.setFont(consoleFont);

consoleArea.setForeground(Color.*WHITE*);

JScrollPane consolePane = **new** JScrollPane(consoleArea);

add(consolePane);

corbato = **null**;

step = 0;

}

}

**private** **class** WaitTimePanel **extends** JPanel {

**private** Corbato corbato;

**private** GraphPanel graph;

**private** JTextField maxLambdaField;

**private** JTextField minLambdaField;

**private** JTextField lambdaStepField;

**private** JTextField stepCountField;

**public** WaitTimePanel() {

setLayout(**new** BorderLayout());

JPanel optionsPanel = **new** JPanel();

optionsPanel.setLayout(**new** GridLayout(5, 1));

JPanel tempPanel = **new** JPanel();

minLambdaField = **new** JTextField();

minLambdaField.setColumns(10);

tempPanel.add(**new** JLabel("Минимальная интенсивность входного потока заявок"));

tempPanel.add(minLambdaField);

optionsPanel.add(tempPanel);

tempPanel = **new** JPanel();

maxLambdaField = **new** JTextField();

maxLambdaField.setColumns(10);

tempPanel.add(**new** JLabel("Максимальная интенсивность входного потока заявок"));

tempPanel.add(maxLambdaField);

optionsPanel.add(tempPanel);

tempPanel = **new** JPanel();

lambdaStepField = **new** JTextField();

lambdaStepField.setColumns(10);

tempPanel.add(**new** JLabel("Шаг изменения интенсивности"));

tempPanel.add(lambdaStepField);

optionsPanel.add(tempPanel);

tempPanel = **new** JPanel();

stepCountField = **new** JTextField();

stepCountField.setColumns(10);

tempPanel.add(**new** JLabel("Количество шагов"));

tempPanel.add(stepCountField);

optionsPanel.add(tempPanel);

JButton generateButton = **new** JButton();

generateButton.setAction(**new** AbstractAction() {

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

**double** intensity = Double.*valueOf*(minLambdaField.getText());

graph.setModel(**new** GraphModel("График среднего времени ожидания от интенсивности",

"Интенсивность", "Среднее время ожидания"));

**while** (intensity <= Double.*valueOf*(maxLambdaField.getText())) {

corbato = **new** Corbato(10, 10, 100, intensity, 1, 10000);

**for** (**int** i = 0; i < Integer.*valueOf*(stepCountField.getText()); i++) {

corbato.tact();

}

graph.getModel().add(intensity, corbato.getAverageWaitTime());

intensity += Double.*valueOf*(lambdaStepField.getText());

}

}

});

generateButton.setText("Построить");

tempPanel = **new** JPanel();

tempPanel.add(generateButton);

optionsPanel.add(tempPanel);

add(optionsPanel, BorderLayout.*NORTH*);

graph = **new** GraphPanel();

graph.setModel(**new** GraphModel("График среднего времени ожидания от интенсивности",

"Интенсивность", "Среднее время ожидания"));

add(graph);

}

}

**private** **class** CPUDowntimePanel **extends** JPanel {

**private** Corbato corbato;

**private** GraphPanel graph;

**private** JTextField maxLambdaField;

**private** JTextField minLambdaField;

**private** JTextField lambdaStepField;

**private** JTextField stepCountField;

**public** CPUDowntimePanel() {

setLayout(**new** BorderLayout());

JPanel optionsPanel = **new** JPanel();

optionsPanel.setLayout(**new** GridLayout(5, 1));

JPanel tempPanel = **new** JPanel();

minLambdaField = **new** JTextField();

minLambdaField.setColumns(10);

tempPanel.add(**new** JLabel("Минимальная интенсивность входного потока заявок"));

tempPanel.add(minLambdaField);

optionsPanel.add(tempPanel);

tempPanel = **new** JPanel();

maxLambdaField = **new** JTextField();

maxLambdaField.setColumns(10);

tempPanel.add(**new** JLabel("Максимальная интенсивность входного потока заявок"));

tempPanel.add(maxLambdaField);

optionsPanel.add(tempPanel);

tempPanel = **new** JPanel();

lambdaStepField = **new** JTextField();

lambdaStepField.setColumns(10);

tempPanel.add(**new** JLabel("Шаг изменения интенсивности"));

tempPanel.add(lambdaStepField);

optionsPanel.add(tempPanel);

tempPanel = **new** JPanel();

stepCountField = **new** JTextField();

stepCountField.setColumns(10);

tempPanel.add(**new** JLabel("Количество шагов"));

tempPanel.add(stepCountField);

optionsPanel.add(tempPanel);

JButton generateButton = **new** JButton();

generateButton.setAction(**new** AbstractAction() {

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

**double** intensity = Double.*valueOf*(minLambdaField.getText());

graph.setModel(**new** GraphModel("График процента простоя ресурса от интенсивности",

"Интенсивность", "Процент простоя ресурса"));

**while** (intensity <= Double.*valueOf*(maxLambdaField.getText())) {

corbato = **new** Corbato(10, 10, 100, intensity, 1, 10000);

**for** (**int** i = 0; i < Integer.*valueOf*(stepCountField.getText()); i++) {

corbato.tact();

}

graph.getModel().add(intensity, corbato.getCPUdowntimePercent());

intensity += Double.*valueOf*(lambdaStepField.getText());

}

}

});

generateButton.setText("Построить");

tempPanel = **new** JPanel();

tempPanel.add(generateButton);

optionsPanel.add(tempPanel);

add(optionsPanel, BorderLayout.*NORTH*);

graph = **new** GraphPanel();

graph.setModel(**new** GraphModel("График процента простоя ресурса от интенсивности",

"Интенсивность", "Процент простоя ресурса"));

add(graph);

}

}

**private** **class** PriorityPanel **extends** JPanel {

**private** Corbato corbato;

**private** GraphPanel graph;

**private** JTextField lambdaField;

**private** JTextField stepCountField;

**public** PriorityPanel() {

setLayout(**new** BorderLayout());

JPanel optionsPanel = **new** JPanel();

optionsPanel.setLayout(**new** GridLayout(3, 1));

JPanel tempPanel = **new** JPanel();

lambdaField = **new** JTextField();

lambdaField.setColumns(10);

tempPanel.add(**new** JLabel("Интенсивность входного потока заявок"));

tempPanel.add(lambdaField);

optionsPanel.add(tempPanel);

tempPanel = **new** JPanel();

stepCountField = **new** JTextField();

stepCountField.setColumns(10);

tempPanel.add(**new** JLabel("Количество шагов"));

tempPanel.add(stepCountField);

optionsPanel.add(tempPanel);

JButton generateButton = **new** JButton();

generateButton.setAction(**new** AbstractAction() {

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

**double** intensity = Double.*valueOf*(lambdaField.getText());

graph.setModel(**new** GraphModel("График среднего времени ожидания от приоритета",

"Приоритет заявки", "Среднее время ожидания"));

**for** (**int** priority = 1; priority < 10000; priority++) {

corbato = **new** Corbato(10, 10, 100, intensity, priority, priority);

**for** (**int** i = 0; i < Integer.*valueOf*(stepCountField.getText()); i++) {

corbato.tact();

}

graph.getModel().add(priority, corbato.getAverageWaitTime());

}

}

});

generateButton.setText("Построить");

tempPanel = **new** JPanel();

tempPanel.add(generateButton);

optionsPanel.add(tempPanel);

add(optionsPanel, BorderLayout.*NORTH*);

graph = **new** GraphPanel();

graph.setModel(**new** GraphModel("График среднего времени ожидания от приоритета",

"Приоритет заявки", "Среднее время ожидания"));

add(graph);

}

}

}

**package** os.lab3;

**import** org.jfree.chart.ChartFactory;

**import** org.jfree.chart.ChartPanel;

**import** org.jfree.chart.JFreeChart;

**import** org.jfree.chart.plot.PlotOrientation;

**import** org.jfree.chart.plot.XYPlot;

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

**import** java.awt.\*;

**public** **class** GraphPanel **extends** JPanel {

**private** JFreeChart chart;

**private** GraphModel model;

**public** GraphPanel() {

**super**();

}

**public** **void** setModel(GraphModel model) {

**this**.model = model;

refresh();

}

**public** GraphModel getModel() {

**return** model;

}

**public** **void** refresh() {

chart = ChartFactory.*createXYLineChart*(

**this**.model.getName(),

**this**.model.getXName(),

**this**.model.getYName(),

**this**.model.getDataset(),

PlotOrientation.*VERTICAL*,

**false**,

**false**,

**false**

);

XYPlot plot = chart.getXYPlot();

plot.setBackgroundPaint(Color.*DARK\_GRAY*);

plot.setDomainGridlinePaint(Color.*WHITE*);

plot.setRangeGridlinePaint(Color.*WHITE*);

**this**.removeAll();

add(**new** ChartPanel(chart));

revalidate();

repaint();

}

}

**package** os.lab3;

**import** org.jfree.data.xy.XYSeries;

**import** org.jfree.data.xy.XYSeriesCollection;

**import** java.util.HashMap;

**public** **class** GraphModel {

**private** String name;

**private** String xName;

**private** String yName;

**private** XYSeries data;

**private** HashMap<Double, Integer> xCount;

**public** GraphModel(String name, String xName, String yName) {

**this**.name = name;

**this**.xName = xName;

**this**.yName = yName;

data = **new** XYSeries(**this**.name, **true**);

xCount = **new** HashMap<Double, Integer>();

}

**public** String getName() {

**return** name;

}

**public** String getXName() {

**return** xName;

}

**public** String getYName() {

**return** yName;

}

**public** XYSeriesCollection getDataset() {

**return** **new** XYSeriesCollection(data);

}

**public** **void** removeAll() {

data = **new** XYSeries(**this**.name, **true**);

xCount = **new** HashMap<Double, Integer>();

}

**public** **void** add(**double** x, **double** y) {

**if** (xCount.containsKey(Double.*valueOf*(x))) {

**int** i = 0;

**while** (data.getX(i).doubleValue() != x) {

i++;

}

**int** prevCount = xCount.get(Double.*valueOf*(x));

xCount.put(x, Integer.*valueOf*(++prevCount));

**double** temp = data.getY(i).doubleValue();

temp += y;

temp /= xCount.get(Double.*valueOf*(x).doubleValue());

data.remove(i);

data.add(x, temp);

}

**else** {

data.add(x, y);

xCount.put(Double.*valueOf*(x), 1);

}

}

}

**package** os.lab3;

**import** java.util.ArrayList;

**public** **class** Corbato {

**private** **int** tk;

**private** **int** ltk;

**private** ArrayList<ProgramQueue> queues;

**private** **double** lambda;

**private** **int** minLp;

**private** **int** maxLp;

**private** **int** t;

**private** **int** generateCount;

**private** **double** averageWaitTime;

**private** **double** sumWaitTime;

**private** **int** doneCount;

**private** **int** CPUdowntime;

**public** Corbato(**int** queuesCount, **int** tk, **int** ltk, **double** lambda, **int** minLp, **int** maxLp) {

queues = **new** ArrayList<ProgramQueue>();

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

queues.add(**new** ProgramQueue());

}

**this**.tk = tk;

**this**.ltk = ltk;

**this**.lambda = lambda;

**this**.minLp = minLp;

**this**.maxLp = maxLp;

t = 0;

generateCount = 0;

doneCount = 0;

averageWaitTime = 0;

sumWaitTime = 0;

CPUdowntime = 0;

}

**public** **double** getAverageWaitTime() {

**return** averageWaitTime;

}

**public** **double** getCPUdowntimePercent() {

**return** ((**double**) CPUdowntime / (**double**) t);

}

**public** **void** addProgram(Program p) {

**int** i = (**int**) (Math.*log*((**double**) p.getLp() / ltk) / Math.*log*(2));

**if** (++i >= queues.size()) {

i = queues.size() - 1;

}

**if** (i < 0) {

i = 0;

}

queues.get(i).add(p);

}

**public** **void** tact() {

**int** i = 0;

**while** (((**double**) t \* lambda) > generateCount) {

Program p = Program.*generateProgram*(minLp, maxLp);

addProgram(p);

generateCount++;

}

**while** ((i < queues.size()) && (queues.get(i).getSize() == 0)) {

i++;

}

**boolean** isResourceInUse = **false**;

Program program = **null**;

**if** (i < queues.size()) {

**for** (**int** j = 0; j < queues.size(); j++) {

**if** (j != i) {

queues.get(j).waitAll((**int**) (Math.*pow*(2, (**double**) i) \* tk));

}

}

program = queues.get(i).waitAllExcept(0, (**int**) (Math.*pow*(2, (**double**) i) \* tk),

(**int**) (Math.*pow*(2, (**double**) i) \* ltk));

**if** (program.getLp() <= 0) {

sumWaitTime += program.getWaitTime();

averageWaitTime = sumWaitTime / ++doneCount;

}

**if** (program.getLp() >= 0) {

**if** ((i + 1) < queues.size()) {

queues.get(i + 1).add(program);

}

**else** {

queues.get(i).add(program);

}

isResourceInUse = **true**;

}

**else** {

isResourceInUse = **false**;

}

}

**if** (isResourceInUse) {

t += (**int**) (Math.*pow*(2, (**double**) i) \* tk);

}

**else** {

**if** (program == **null**) {

t += tk;

CPUdowntime += tk;

}

**else** {

t += (**int**) (Math.*pow*(2, (**double**) i) \* tk);

CPUdowntime += ((**double**) Math.*abs*(program.getLp()) / ltk) \* tk;

}

}

}

**public** String tactWithLog() {

String log = "";

**int** i = 0;

**while** (((**double**) t \* lambda) > generateCount) {

Program p = Program.*generateProgram*(minLp, maxLp);

addProgram(p);

generateCount++;

log += "Добавлена программа с длиной Lp = " + String.*valueOf*(p.getLp()) + "\n";

}

**while** ((i < queues.size()) && (queues.get(i).getSize() == 0)) {

i++;

}

**boolean** isResourceInUse = **false**;

Program program = **null**;

**if** (i < queues.size()) {

**for** (**int** j = 0; j < queues.size(); j++) {

**if** (j != i) {

queues.get(j).waitAll((**int**) (Math.*pow*(2, (**double**) i) \* tk));

}

}

log += "Выполнение программы из очереди №" + String.*valueOf*(i) + " с длиной Lp = " +

String.*valueOf*(queues.get(i).get(0).getLp()) + " - " +

String.*valueOf*((**int**) (Math.*pow*(2, (**double**) i) \* ltk));

program = queues.get(i).waitAllExcept(0, (**int**) (Math.*pow*(2, (**double**) i) \* tk),

(**int**) (Math.*pow*(2, (**double**) i) \* ltk));

**if** (program.getLp() <= 0) {

sumWaitTime += program.getWaitTime();

averageWaitTime = sumWaitTime / ++doneCount;

}

**if** (program.getLp() >= 0) {

**if** ((i + 1) < queues.size()) {

queues.get(i + 1).add(program);

log += " = " + String.*valueOf*(program.getLp()) + "\n";

}

**else** {

queues.get(i).add(program);

log += " = " + String.*valueOf*(program.getLp()) + "\n";

}

isResourceInUse = **true**;

}

**else** {

log += " = " + "0" + "\n";

isResourceInUse = **false**;

}

}

**if** (isResourceInUse) {

t += (**int**) (Math.*pow*(2, (**double**) i) \* tk);

log += "Ресурс занят\n";

}

**else** {

log += "Ресурс свободен\n";

**if** (program == **null**) {

CPUdowntime += tk;

t += tk;

}

**else** {

CPUdowntime += ((**double**) Math.*abs*(program.getLp()) / ltk) \* tk;

t += (**int**) (Math.*pow*(2, (**double**) i) \* tk);

}

}

log += "Системное время: " + String.*valueOf*(t) + "\n";

**return** log;

}

}

**package** os.lab3;

**import** java.util.ArrayList;

**public** **class** ProgramQueue {

**private** ArrayList<Program> programs;

**public** ProgramQueue() {

programs = **new** ArrayList<Program>();

}

**public** **int** getSize() {

**return** programs.size();

}

**public** **void** add(Program p) {

programs.add(p);

}

**public** Program get(**int** i) {

**return** programs.get(i);

}

**public** **void** waitAll(**int** tk) {

**if** (programs.size() > 0) {

**for** (Program p : programs) {

p.wait(tk);

}

}

}

**public** Program waitAllExcept(**int** e, **int** tk, **int** ltk) {

Program program = **null**;

**for** (**int** i = 0; i < programs.size(); i++) {

**if** (i != e) {

programs.get(i).wait(tk);

}

**else** {

programs.get(i).execute(ltk);

program = programs.get(i).copy();

programs.remove(i);

}

}

**return** program;

}

}

**package** os.lab3;

**public** **class** Program {

**private** **int** startLp;

**private** **int** lp;

**private** **int** waitTime;

**public** Program(**int** lp) {

**this**.lp = lp;

startLp = lp;

waitTime = 0;

}

**public** **static** Program generateProgram(**int** minLp, **int** maxLp) {

**return** **new** Program(minLp + (**int**)(Math.*random*() \* (**double**)(maxLp - minLp)));

}

**public** Program copy() {

Program copyProgram = **new** Program(**this**.lp);

copyProgram.waitTime = **this**.waitTime;

copyProgram.startLp = startLp;

**return** copyProgram;

}

**public** **int** getStartLp() {

**return** startLp;

}

**public** **int** getLp() {

**return** lp;

}

**public** **void** execute(**int** ltk) {

lp -= ltk;

}

**public** **void** wait(**int** tk) {

waitTime += tk;

}

**public** **int** getWaitTime() {

**return** waitTime;

}

}

**Графики**

