Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет прикладної математики

Кафедра спеціалізованих комп’ютерних систем

**Лабораторна робота № 3**

З дисципліни

**«Системне програмне забезпечення»**

Виконав

Студент групи КВ-92

Степанюк Михайло

**Київ 2012**

**Постановка задачі:**

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 систем без пріоритетів побудувати графік дисперсії часу очікування при фіксованій інтенсивності вхідного потоку заявок.
5. Пояснити форму графіків.

### *MDRR (Modified Deficit Round Robin )*

Модифікований алгоритм кругового обслуговування з дефіцитом (Modified Deficit Round Robin) використовується для управління переповненнями. Дозволяє організовувати буферизацію трафіку чутливого до затримок у разі використання черги строгого пріоритету.

MDRR класифікує пакети в чергах, базуючись на старших бітах в заголовку IP. Ці черги обслуговуються за карусельним алгоритмом, з чергою, котрій дозволяється передавати ряд байтів кожного разу, коли вона обслуговується (крок квантування). Лічильник дефіциту слідкує, скільки байтів черга передала в кожному проході алгоритму.

**Код програми:**

**dispatcher.h**

#ifndef DISPATCHER\_H

#define DISPATCHER\_H

#include <QObject>

#include <QList>

class Process

{

public:

Process():id(-1), timeArrival(0), timeExecution(0), timeNeeded(0), timeFinished(-1), timeDelay(0){}

Process(int setId, int setArrival, int setExecution):id(setId), timeArrival(setArrival), timeExecution(setExecution), timeNeeded(setExecution), timeFinished(-1), timeDelay(0){}

void onTick(int tick)

{

if(isFinished())

return;

timeNeeded--;

if(isFinished())

timeFinished = tick;

}

void onWait(){timeDelay++;}

int getTimeNeeded(bool total = false) const

{

if(total)

return timeExecution;

else

return timeNeeded;

}

bool isFinished() const {return !(getTimeNeeded() > 0);}

bool isExecutable(int tick) const {return (tick >= timeArrival) && (!isFinished());}

int getId() const {return id;}

int getArrival() const {return timeArrival;}

int getTimeFinished() const {return timeFinished;}

int getTimeDelay() const {return timeDelay;}

protected:

int id;

int timeArrival;

int timeExecution;

int timeNeeded;

int timeFinished;

int timeDelay;

};

class Dispatcher : public QObject

{

Q\_OBJECT

public:

explicit Dispatcher(QObject \* parent = 0);

int getCurrentTime() const {return timeCurrent;}

QList< QList<Process>>& getQueueRR() {return queueRR;}

QList<int>& getDeltas() {return deltas;}

bool getAllReady(){return allReady;}

int getDoNothingTicks(){return do\_nothing;}

protected:

QList< QList<Process>> queueRR;

QList<int> deltas;

int do\_nothing;

int timeCurrent;

QList<int> idRR;

QList<int> idRRChange;

bool allReady;

bool onTickRR(int q, int timeDelta);

signals:

void onFinish();

void onExecute(int queue, int index);

public slots:

void onTick();

void onReset();

void onPrepare();

};

#endif // DISPATCHER\_H

**dispatcher.cpp**

#include "dispatcher.h"

Dispatcher::Dispatcher(QObject \* parent):QObject(parent){onReset();}

bool Dispatcher::onTickRR(int q, inttimeDelta)

{

if(idRRChange[q] == 0)

{

idRRChange[q] = timeDelta;

boolsetuped = false;

for(int i = 0; i <queueRR[q].count(); ++i)

{

if((i >idRR[q]) &&queueRR[q][i].isExecutable(getCurrentTime()))

{

idRR[q] = i;

setuped = true;

break;

}

}

if(!setuped)

{

for(int i = 0; i <queueRR[q].count(); ++i)

if(queueRR[q][i].isExecutable(getCurrentTime()))

{

idRR[q] = i;

setuped = true;

break;

}

}

if(!setuped)

idRR[q] = -1;

}

for(int i = 0; i <queueRR[q].count(); ++i)

{

if((i == idRR[q]) &&queueRR[q][i].isExecutable(getCurrentTime()))

queueRR[q][i].onTick(getCurrentTime());

else if(queueRR[q][i].isExecutable(getCurrentTime()))

queueRR[q][i].onWait();

}

idRRChange[q]--;

if((idRR[q] != -1) && !queueRR[q][idRR[q]].isExecutable(getCurrentTime()))

idRRChange[q] = 0;

if(idRR[q] == -1)

idRRChange[q] = 0;

onExecute(q, idRR[q]);

returnidRR[q] != -1;

}

void Dispatcher::onTick()

{

inttotalSum = 0;

foreach(int delta, deltas)

totalSum += delta;

QList<int> credits = deltas;

for(int i = 0; i <totalSum; ++i)

{

QList<int>credits\_temp = credits;

while(true)

{

if(!credits\_temp.size()) // no executable queues

break;

bool failed = true;

for(int j = 0; j <credits\_temp.size(); ++j)

if(credits\_temp[j] != -1)

failed = false;

if(failed)

{

do\_nothing++;

break;

}

intbestq = 0;

for(int j = 0; j <credits\_temp.size(); ++j)

if(credits\_temp[j] >credits\_temp[bestq])

bestq = j; //find best executable

credits\_temp[bestq] = -1; // disable it for futher search

if(onTickRR(bestq, 1)) // get one time quant to int

{

for(int j = 0; j <queueRR.size(); ++j)

{

if(j != bestq)

{

for(int j2 = 0; j2 <queueRR[j].count(); ++j2)

{

if(queueRR[j][j2].isExecutable(getCurrentTime()))

queueRR[j][j2].onWait();

}

}

}

credits[bestq]--;

break;

}

}

allReady = true;

for(int i = 0; i <queueRR.size(); ++i)

{

foreach(Process process, queueRR[i])

if(!process.isFinished())

{

allReady = false;

break;

}

}

if(allReady)

{

onFinish();

return;

}

else

timeCurrent++;

}

}

void Dispatcher::onReset()

{

queueRR.clear();

deltas.clear();

do\_nothing = 0;

timeCurrent = 0;

idRR.clear();

idRRChange.clear();

allReady = false;

}

boolcaseSortProcess(const Process & p1, const Process & p2){return p1.getArrival() < p2.getArrival();}

void Dispatcher::onPrepare()

{

foreach(int temp, deltas)

{

idRR.push\_back(-1);

idRRChange.push\_back(0);

}

for(int i = 0; i <queueRR.size(); ++i)

qSort(queueRR[i].begin(), queueRR[i].end(), caseSortProcess);

}

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "dispatcher.h"

namespaceUi {class MainWindow;}

classMainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicitMainWindow(QWidget \*parent = 0);

~MainWindow();

public slots:

voidonFinished();

voidlololo();

private slots:

voidon\_actionQuit\_triggered();

voidon\_queues\_valueChanged(int );

voidon\_pushButton\_clicked();

voidon\_webView\_loadFinished(bool );

void on\_pushButton\_2\_clicked();

private:

Ui::MainWindow \*ui;

QStringjscode;

floatprob;

floatprobFrom;

floatprobTo;

floatprobDelta;

intlenFrom;

intlenTo;

intlenTotal;

QVector<int> quants;

QVector<float>av\_wait, av\_dn;

Dispatcher dispatcher;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QWebFrame>

#include <QMessageBox>

#include <QCoreApplication>

#include <QTimer>

#include "dialog.h"

#include "ui\_dialog.h"

MainWindow::MainWindow(QWidget \*parent):QMainWindow(parent), ui(new Ui::MainWindow)

{

connect(&dispatcher, SIGNAL(onFinish()), this, SLOT(onFinished()));

ui->setupUi(this);

ui->webView->load(QUrl("qrc:///index.html"));

ui->webView->show();

ui->probFrom->setText("0.01");

ui->probTo->setText("0.2");

ui->probDelta->setText("0.01");

ui->lenFrom->setText("6");

ui->lenTo->setText("10");

ui->lenTotal->setText("1000");

ui->progressBar->setValue(0);

ui->queues->setValue(3);

}

MainWindow::~MainWindow(){delete ui;}

voidMainWindow::on\_actionQuit\_triggered(){ close();}

voidMainWindow::on\_queues\_valueChanged(int v)

{

ui->quants->clear();

for(int i = 0; i < v; ++i)

ui->quants->append("100");

}

voidMainWindow::on\_pushButton\_clicked()

{

probFrom = ui->probFrom->text().toFloat();

probTo = ui->probTo->text().toFloat();

probDelta = ui->probDelta->text().toFloat();

lenFrom = ui->lenFrom->text().toInt();

lenTo = ui->lenTo->text().toInt();

lenTotal = ui->lenTotal->text().toInt();

QStringquantsText = ui->quants->document()->toPlainText();

QString temp;

QStringListeee = quantsText.split("\n", QString::SkipEmptyParts);

quants.clear();

foreach(temp, eee)

quants.push\_back(temp.toInt());

av\_wait.clear();

av\_dn.clear();

ui->progressBar->setValue(0);

prob = probFrom;

QTimer::singleShot(0, this, SLOT(lololo()));

}

voidMainWindow::onFinished(){}

voidMainWindow::lololo()

{

if(prob<= probTo)

{

dispatcher.onReset();

for(int i = 0; i <quants.size(); ++i)

{

dispatcher.getQueueRR().push\_back(QList<Process>());

dispatcher.getDeltas().push\_back(quants[i]);

}

int id = 0;

for(intttt = 0; ttt<lenTotal; ++ttt)

for(int i = 0; i <quants.size(); ++i)

if(((float)rand() / (float)RAND\_MAX) <prob)

dispatcher.getQueueRR()[i].push\_back(Process(id++, ttt, rand()%(lenFrom - lenTo + 1) + lenTo));

dispatcher.onPrepare();

while(!dispatcher.getAllReady())

dispatcher.onTick();

intwait\_sum = 0, wait\_count = 0;

intexec\_sum = 0, exec\_count = 0;

for(int i = 0; i <dispatcher.getQueueRR().size(); ++i)

{

foreach(Process temp, dispatcher.getQueueRR()[i])

{

wait\_sum += temp.getTimeDelay();

wait\_count ++;

//exec\_sum += temp

}

}

av\_wait.push\_back((float)wait\_sum / (float)wait\_count);

if(dispatcher.getDoNothingTicks() >lenTotal)

av\_dn.push\_back(1.0f);

else

av\_dn.push\_back((float)dispatcher.getDoNothingTicks() / (float)lenTotal);

ui->progressBar->setValue(100 \* ((prob - probFrom) / (probTo - probFrom)));

prob += probDelta;

QTimer::singleShot(0, this, SLOT(lololo()));

}

else

{

ui->progressBar->setValue(100);

jscode = "";

jscode+="var mm1 = new JSChart('graph', 'line');";

jscode+="mm1.setDataArray([";

intwwi = 0;

for(float prob = probFrom; prob<= probTo; prob += probDelta)

{

jscode+=QString("[%1, %2],").arg(prob).arg(av\_wait[wwi]);

wwi++;

}

jscode.remove(jscode.size() - 1, 1);

jscode+="], 'blue');";

jscode+="mm1.setBackgroundImage('chart\_bg.jpg');";

jscode+="mm1.setTitle('average wait');";

jscode+="mm1.setSize(616, 321);";

jscode+="mm1.draw();";

jscode+="var mm2 = new JSChart('graph2', 'line');";

jscode+="mm2.setDataArray([";

int wwi2 = 0;

for(float prob = probFrom; prob<= probTo; prob += probDelta)

{

jscode+=QString("[%1, %2],").arg(prob).arg((int)(av\_dn[wwi2] \* 100.0f));

wwi2++;

}

jscode.remove(jscode.size() - 1, 1);

jscode+="], 'green');";

jscode+="mm2.setBackgroundImage('chart\_bg.jpg');";

jscode+="mm2.setLineColor('#A4D314', 'green');";

jscode+="mm2.setTitle('system wait / run time');";

jscode+="mm2.setSize(616, 321);";

jscode+="mm2.draw();";

ui->webView->reload();

}

}

voidMainWindow::on\_webView\_loadFinished(bool )

{

ui->webView->page()->mainFrame()->evaluateJavaScript(jscode);

}

voidMainWindow::on\_pushButton\_2\_clicked()

{

Dialog dial(this);

dial.ui->processTable->clear();

intsizetotal=0;

for(int i = 0; i <dispatcher.getQueueRR().size(); ++i)

sizetotal += dispatcher.getQueueRR()[i].size();

dial.ui->processTable->setRowCount(sizetotal);

dial.ui->processTable->setColumnCount(6);

dial.ui->processTable->setHorizontalHeaderLabels(QStringList() << "queue" << "t arrival" << "t execute" << "t needed" << "t finish" << "t delay");

int row = 0;

for(int i = 0; i <dispatcher.getQueueRR().size(); ++i)

foreach(Process process, dispatcher.getQueueRR()[i])

{

dial.ui->processTable->setItem(row, 0, new QTableWidgetItem(QString("%1").arg(i)));

dial.ui->processTable->setItem(row, 1, new QTableWidgetItem(QString("%1").arg(process.getArrival())));

dial.ui->processTable->setItem(row, 2, new QTableWidgetItem(QString("%1").arg(process.getTimeNeeded(true))));

dial.ui->processTable->setItem(row, 3, new QTableWidgetItem(QString("%1").arg(process.getTimeNeeded(false))));

dial.ui->processTable->setItem(row, 4, new QTableWidgetItem(QString("%1").arg(process.getTimeFinished())));

dial.ui->processTable->setItem(row, 5, new QTableWidgetItem(QString("%1").arg(process.getTimeDelay())));

row++;

}

dial.exec();

}

