Национальный технический университет Украины
«Киевский политехнический институт»
Факультет информатики и вычислительной техники
Кафедра вычислительной техники

Лабораторная работа №3

По курсу «СПО»

Выполнил:
Студент III курса ФИВТ
Группы IO-93
Свинарчук Сергей

Киев – 2012

№$_{зк}$ 9316 => 16 mod 16+1 = 1

Вариант: Алгоритм FIFO (*First Input- First Output*) з пріоритетами, без витіснення

First in-first out (fifo)

Алгоритм обслуживания очередей firstin, firstout (fifo), также называемый first come first served является простым.

Fifo является наиболее простой стратегией планирования процессов и заключается в том, что ресурс передается тому процессу, который раньше всех остальных обратился к нему. Когда процесс попадает в очередь готовых процессов, process control block присоединяется к хвосту очереди. Среднее время ожидания для стратегии fifo часто является достаточно большим и зависит от порядка поступления процессов в очередь готовых процессов.

Стратегии fifo присущ так называемый "эффект конвоя". В том случае, когда в компьютере есть один большой процесс и несколько малых, то все процессы собираются в начале очереди готовых процессов, а затем в очереди к оборудованию. Таким образом, "эффект конвоя" приводит к снижению пропускной способности как процессора, так и периферийного устройства.

Дисциплина обслуживания fifo с приоритетами без вытеснения предполагает, что каждая заявка имеет свой приоритет. Заявки с одинаковыми приоритетами группируются в очередь типа fifo. Сначала обслуживается очередь с высшим приоритетом. Заявка, попавшая в процессор не может быть вытеснена из него пока не завершится ее обслуживания.

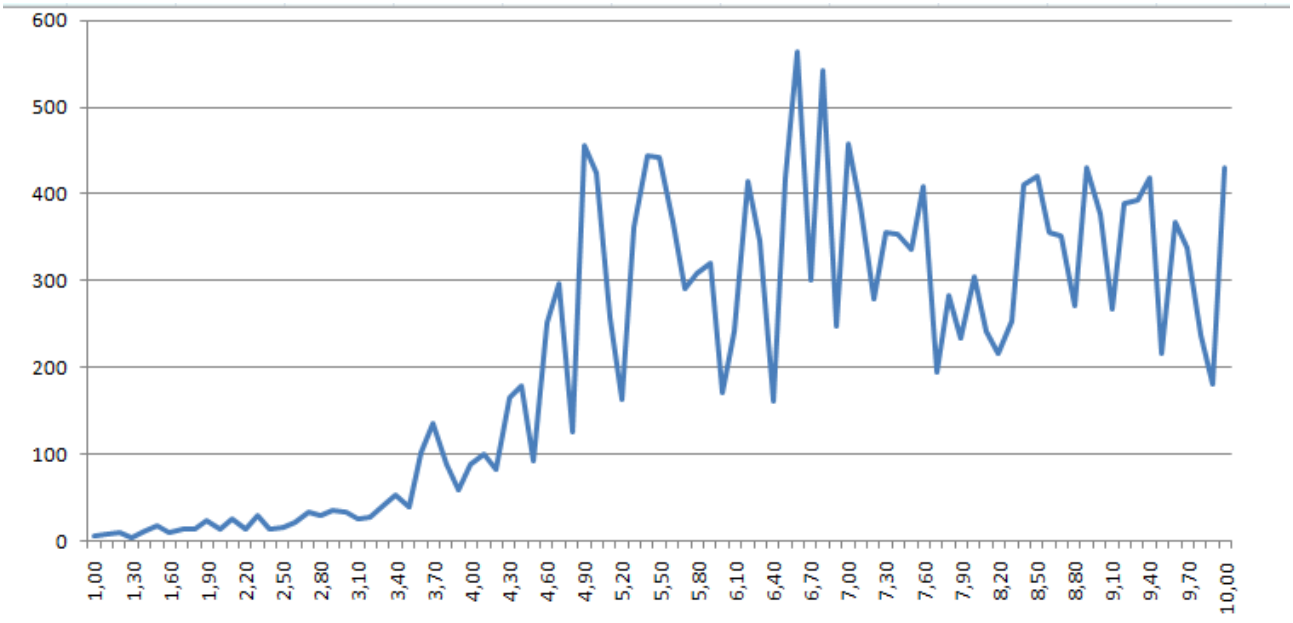


График зависимости среднего время ожидания заявки в очереди от интенсивности входящего потока

Как видно, при увеличении интенсивности заявки ждут больше времени в очереди. Это означает что они поступают быстрей чем успевают обрабатываться. Хотя зависимость и нелинейная, но перестает быстро расти уже при значениях интенсивности более 5.

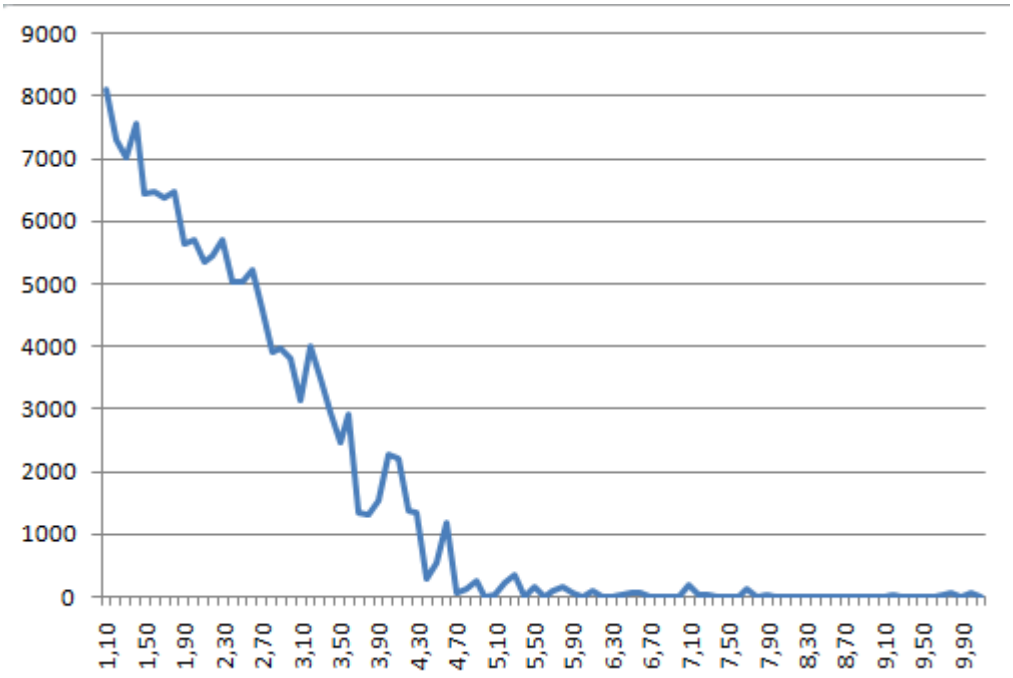


График зависимости времени простоя от интенсивности потока:

Уже при интенсивности в 4,7, процессор практически перестает простаивать и процент времени без нагрузки стремится к нулю.

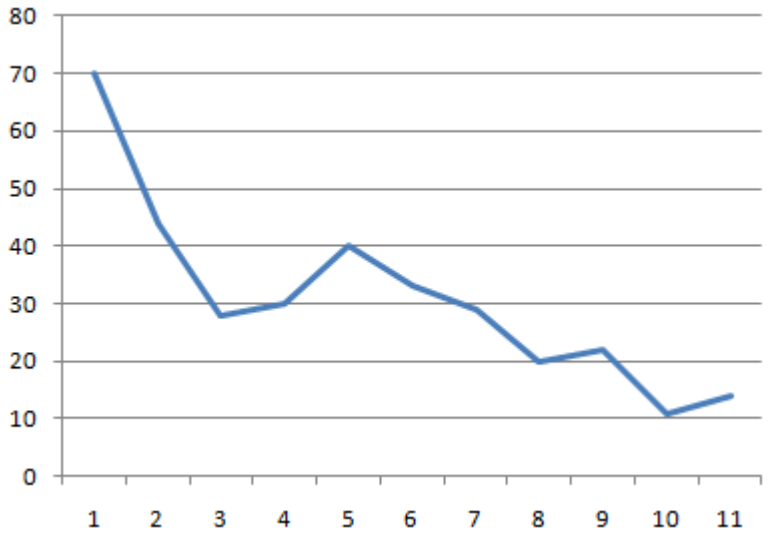


График зависимости среднего времени ожидания от приоритета задачи:

С графика видна тенденция, что при увеличении приоритета поступающей задачи время ожидания её обработки уменшаеться.

Код программы

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 * User: vaifer
 * Date: 17.05.12
 * Time: 21:26
 * To change this template use File | Settings | File Templates.
 */
public class Processor {
    //массив очередей
    ArrayList<ArrayList<Request>> quenes = new ArrayList<ArrayList<Request>>();
    //массив обслуженых заявок
    ArrayList<Request> readyRequest = new ArrayList<Request>();
    //Свободен или занят процессор
    boolean procStatus = true;
    //Заявка, что обрабатывается
    Request r;
    //время простоя
    int stopTime;
    boolean stopStatus = true;
    //максимальный приоритет
    int Prioruty_max = 11;
    //Среднее время ожижания по приоритетам
    int[] timePriority = new int[Prioruty_max];
    int[] counter = new int[Prioruty_max];
    //добавление новой заявки в очередь с тем самым приоритетом
    public void addRequest(int id, int lamda, int time){
        Generator generator = new Generator();
        int timeService = 0;
        while(true){
            if(timeService < 1){
                timeService = generator.generateExponential(lamda);
            } else{
                break;
            }
        }
        int priority = 0;
        while(true){
            if(priority > 11 || priority < 1){
                priority = generator.generateExponential(lamda);
            } else{
                break;
            }
        }
//        System.out.println("id "+id+"    priority "+priority+"    time service "+timeService);
        Request request = new Request(priority, id, timeService, time);
        boolean status = false;
        for(int i = 0; i < quenes.size(); i++){
            if(quenes.get(i).get(0).getPriority() == request.getPriority()){
                quenes.get(i).add(request);
                status = true;
            }
        }
        if(status){
            return;
        } else {
            addNewLine(request);
        }
```

```
        }
        //добавление с новым приоритетом
        public void addNewLine(Request request){
            quenes.add(new ArrayList<Request>());
            quenes.get(quenes.size()-1).add(request);
        }
        //обработка одного цыкла работы
        public void procesRequest(int time){
            int maxPriority = 0;
            int queneMax = 0;
            if(procStatus && !quenes.isEmpty()){
                for(int i = 0; i < quenes.size(); i++){
                    if(maxPriority < quenes.get(i).get(0).getPriority()){
                        maxPriority = quenes.get(i).get(0).getPriority();
                        queneMax = i;

                    }
                }
                r = quenes.get(queneMax).remove(0);
                r.setTimeBeginProc(time);
                if(quenes.get(queneMax).isEmpty()){
                    quenes.remove(queneMax);
                }
                procStatus = false;
                stopStatus = false;
            }

            if(!procStatus && ((time - r.getTimeBeginProc()) >= r.getTimeService())){

                procStatus = true;
                r.setTimeEndProc(time);
                readyRequest.add(r);
                stopStatus = false;
            }
            if(stopStatus && procStatus){
                stopTime++;
            }
            stopStatus = true;
        }
        ////////////////////////////////////////////////////////////////////////////////////////
        ////////////////////          Методы сбора статистической информации //////////////////////
        ////////////////////////////////////////////////////////////////////////////////////////
        public void writeStatictic(double lamda) throws IOException {
            BufferedWriter file = new BufferedWriter(new FileWriter(new
File("Stat_"+Double.toString(lamda)+".csv")));
        //        file.write("ID;Приоритет;время_обслуживания;время_прихода;Начало_обслуживания;Конец
Обслуживания\n");
            for(int i = 0; i < readyRequest.size(); i++){
                file.write(readyRequest.get(i).toString());
        //          System.out.println(readyRequest.get(i).toString());
            }
            if(file != null){
                file.close();
            }
        }

        public int getAvarageTime(){
            int time = 0;
            for(int i = 0; i < readyRequest.size(); i++){
                time += readyRequest.get(i).getTimeBeginProc()-readyRequest.get(i).getTimeCome();
            }
            time /= readyRequest.size();
            time *= 10;
            time = Math.round(time);
            time /= 10;
```

```
            return time;
        }

        public int getStopTime(){
            return stopTime;
        }

        public int[] getTimePriority(){
            System.out.print(timePriority.length);
            for(int i = 0; i < timePriority.length; i++){
                timePriority[i] = 0;
                counter[i] = 0;
            }
            for(int i = 0; i < readyRequest.size(); i++){
                timePriority[readyRequest.get(i).getPriority()-1] += readyRequest.get(i).getTimeBeginProc()-
readyRequest.get(i).getTimeCome();
                counter[readyRequest.get(i).getPriority()-1]++;
            }
            for (int i = 0; i < timePriority.length; i++) {
                if(counter[i] > 0){
                    timePriority[i] /= counter[i];
                }
            }
            return timePriority;
        }
    }

    /**
     * Created by IntelliJ IDEA.
     * User: vaifer
     * Date: 17.05.12
     * Time: 21:20
     * To change this template use File | Settings | File Templates.
     */
    public class Request {
        private int priority;   //приотритет заявки
        private int id;
        private int timeService;//время обработки заявки
        private int timeCome;      //время прихода заявки в очередь
        private int timeBeginProc;   //время начала обработки
        private int timeEndProc;     //время окончание обработки

        public Request(int priority, int id, int timeService, int timeCome){
            this.priority = priority;
            this.id = id;
            this.timeService = timeService;
            this.timeCome = timeCome;
        }

        public int getTimeEndProc() {
            return timeEndProc;
        }

        public void setTimeEndProc(int timeEndProc) {
            this.timeEndProc = timeEndProc;
        }

        public int getTimeBeginProc() {
            return timeBeginProc;
        }

        public void setTimeBeginProc(int timeBeginProc) {
            this.timeBeginProc = timeBeginProc;
        }
```

```
    public int getPriority() {
        return priority;
    }

    public void setPriority(int priority) {
        this.priority = priority;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getTimeService() {
        return timeService;
    }

    public void setTimeService(int timeService) {
        this.timeService = timeService;
    }

    public int getTimeCome() {
        return timeCome;
    }

    public void setTimeCome(int timeCome) {
        this.timeCome = timeCome;
    }

    public String toString(){
        return Integer.toString(id)+";"+Integer.toString(priority)+";"+Integer.toString(timeService)+";"

+Integer.toString(timeCome)+";"+Integer.toString(timeBeginProc)+";"+Integer.toString(timeEndProc)+"\n";
    }
}

/**
 * Created by IntelliJ IDEA.
 * User: vaifer
 * Date: 17.05.12
 * Time: 21:26
 * To change this template use File | Settings | File Templates.
 */
public class Generator {

    public int generateExponential(double lambda) {
        return (int)(-100*Math.log(1-Math.random())/lambda);
    }
}

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Created by IntelliJ IDEA.
 * User: vaifer
 * Date: 17.05.12
 * Time: 21:17
 * To change this template use File | Settings | File Templates.
 */
```

```
public class Test {
    public static void main(String[] args) throws IOException {
        double lamda1 = 10;    //интенсивность вхоодного потока
        int lamda2 = 5;    // время обслуживание заявки
        BufferedWriter file1 = new BufferedWriter(new FileWriter(new File("Graphic_1.csv")));   //График зависимости среднего времени ожидания заявки в очереди от интенсивности входного потока
        BufferedWriter file2 = new BufferedWriter(new FileWriter(new File("Graphic_2.csv")));   //График зависимости времени простоя от интенсивности потока
        BufferedWriter file3 = new BufferedWriter(new FileWriter(new File("Graphic_3.csv")));   //График зависимости среднего времени ожидания от приоритета задачи
        Generator generator = new Generator();
        for(lamda1 = 1; lamda1 <= 10; lamda1+=.1){
            int globalTime = 0;
            int timeNext = 0;
            int id = 1;
            int avarageTime = 0;
            Processor proc = new Processor();
            for( globalTime =0; globalTime < 10000; globalTime++){
                if(timeNext <= 0){
                timeNext = generator.generateExponential(lamda1);
                proc.addRequest(id, lamda2, globalTime);
                id++;
            }
                proc.procesRequest(globalTime);
                timeNext--;
            }
           ////////////////////////////////////////////////////////////////////
          //////    Вивод статистической информации в файлы    /////////
            ////////////////////////////////////////////////////////////////////
            avarageTime = proc.getAvarageTime();
            lamda1 *= 10;
            lamda1 = Math.round(lamda1);
            lamda1 /= 10;
            file1.write(Double.toString(lamda1)+";"+avarageTime+"\n");
            file2.write(Double.toString(lamda1)+";"+proc.getStopTime()+"\n");
            if(lamda1 == 3.0){
                proc.writeStatictic(lamda1);
                for (int i = 0; i < proc.getTimePriority().length; i++){
                    file3.write(Integer.toString(i+1)+";"+Integer.toString(proc.getTimePriority()[i])+"\n");
                }
            }
        }

        file1.close();
        file2.close();
        file3.close();
    }


}
```

Национальный технический университет Украины
«Киевский политехнический институт»
Факультет информатики и вычислительной техники
Кафедра вычислительной техники

Лабораторная работа №3

По курсу «СПО-1»

Выполнил:
Студент III курса ФИВТ
Группы IO-93
Белогородский Владислав

Киев – 2012

Вариант: **смешанный алгоритм**

Каждая из заявок находится в одной из трех очередей обслуживания процессором. Изначально, заявка попадает в первую очередь, в которой она имеет право на три цикла обслуживания по 600 мс, после чего, в случае если заявка остается невыполненной, переходит во вторую очередь, где проходит еще 6 циклов обслуживания. Если и этого не достаточно, то заявка перемещается в третью очередь, где и находится до своего окончательного выполнения. При этом, в каждой из очередей заявки выполняются циклично, а выбор следующей очереди выполняется исходя из приоритетов крайних заявок в очереди. Для генерации времени, необходимого на выполнение заявки, а также входного потока используется генератор чисел, распределенных по экспоненциальному закону.

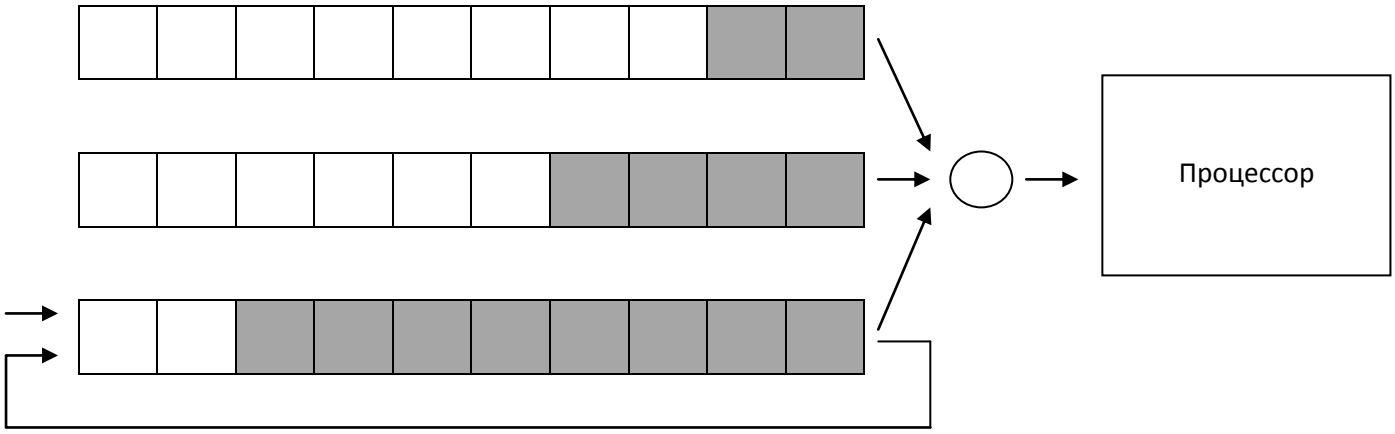
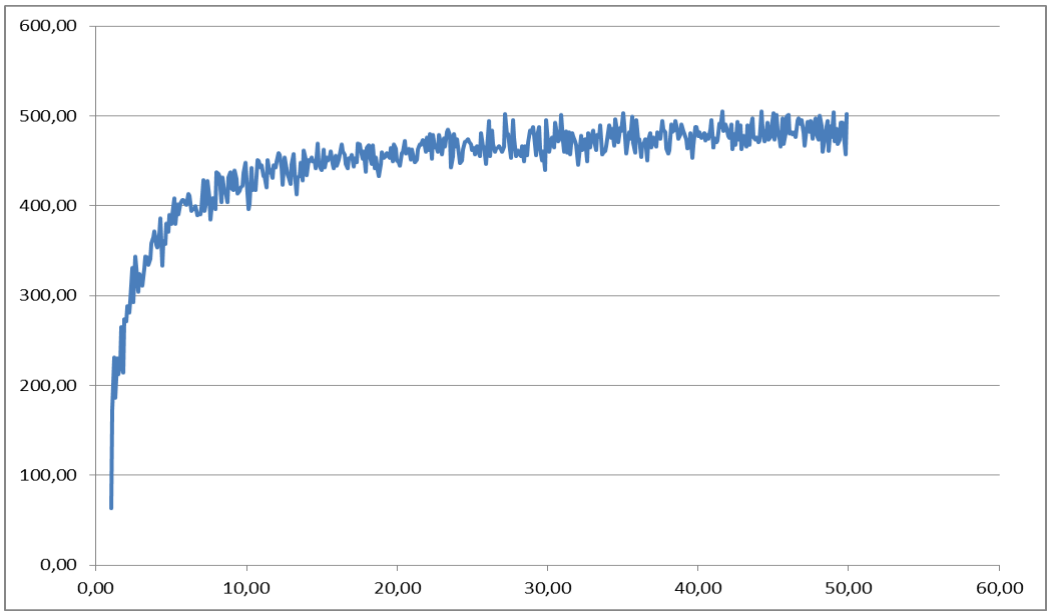
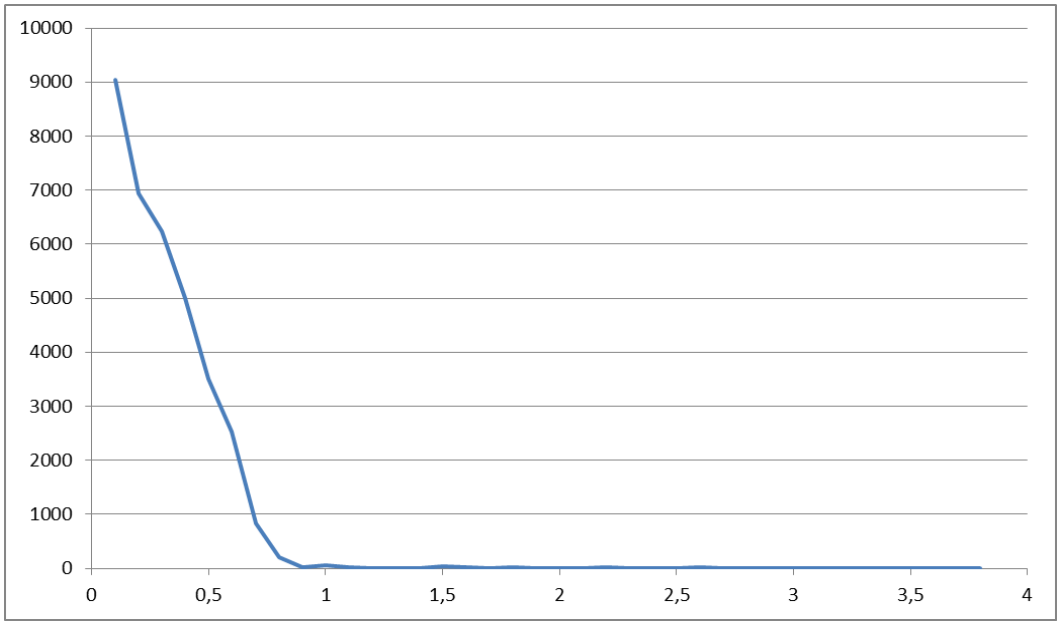


График зависимости среднего времени ожидания заявкой от интенсивности входного потока:



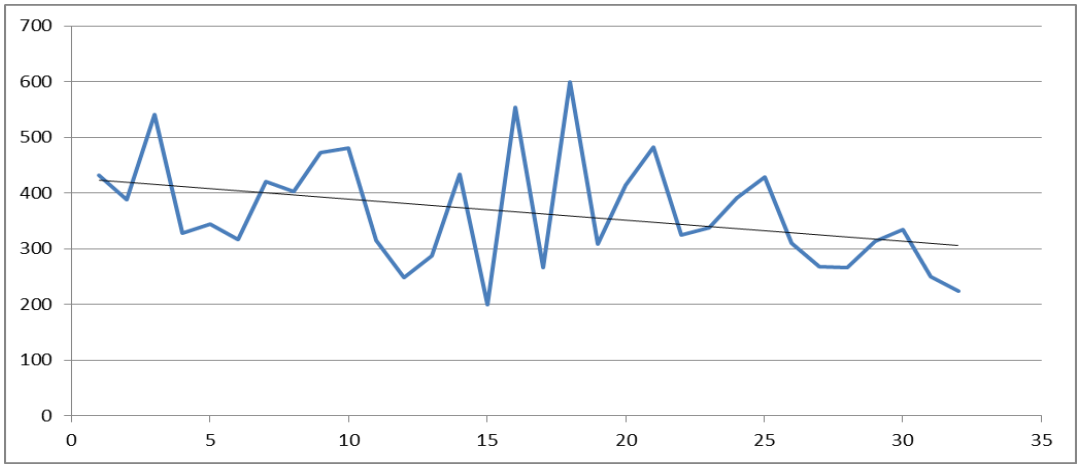
Как видно, при увеличении интенсивности новые заявки поступают чаще, чем процессор успевает их обработать, вследствие чего время ожидания увеличивается, хотя зависимость и нелинейная и перестает быстро расти уже при значениях интенсивности более 20.

График зависимости времени простоя от интенсивности потока:



Уже при интенсивности в 1 (то есть в среднем, одна заявка в секунду), процессор практически перестает простаивать и процент времени без нагрузки стремится к нулю.

График зависимости среднего времени ожидания от приоритета задачи:



Поскольку приоритет задачи учитывается только при выборе потока обслуживания и не играет роль на цикличность обработки заявок в потоке, тенденция уменьшения времени ожидания от роста приоритета наблюдается, но не такая явная

Код программы

```
package lab3;

public class Generator {

	public static double generateExponential(double lambda) {
		return -Math.log(1-Math.random())/lambda;
	}
}
```

```
package lab3;

public class Request {

	private double timeAd;
	private double timeRemove;
	private static int num = 0;
	private int id;
	private double timeForComplete;
	private double firstTime;
	private int attempts;
	private int priority; // from 0 to 32


	public double getTimeForComplete() {
		return timeForComplete;
	}

	public void subTimeForComplete(double time){
		timeForComplete -= time;

	}

	public int getAttempts() {
		return attempts;
	}

	public int getPriority() {
		return priority;
	}

	public void setTimeRemove(double timeRemove) {
		this.timeRemove = timeRemove;
	}

	public void addAttempt(){
		attempts++;
	}

	public double getWaitTime(){
		return timeRemove - timeAd - firstTime;
	}

	public Request(double time, int priority, double timeAd){
		firstTime = timeForComplete = time;
		attempts = 0;
		this.priority = priority;
		this.id = num++;
		this.timeAd = timeAd;
	}
```

```
	@Override
	public String toString() {
		return "Req [id="+id+ ", t=" + Math.round(timeForComplete*1000)/1000.0 +
", ats="
				+ attempts + ", p=" + priority + "]";
	}

}
```

```
package lab3;

import java.util.ArrayList;
import java.util.Random;

public class Processor {

	private ArrayList<Double> waitTimes = new ArrayList<Double>();
	private ArrayList<ArrayList<Request>> queues = new
ArrayList<ArrayList<Request>>();
	private ArrayList<ArrayList<Double>> statistics = new
ArrayList<ArrayList<Double>>();
	private int currentQueue;
	private Random r = new Random();
	private Request current;

	public ArrayList<Double> getWaitTimes() {
		return waitTimes;
	}

	public Processor() {
		for (int i = 0; i < 3; i++) {
			queues.add(new ArrayList<Request>());
		}
		for (int i = 0; i < 33; i++) {
			statistics.add(new ArrayList<Double>());
		}
	}

	public void addNewRequest(double time, double timeAd) {
		queues.get(0).add(new Request(time, r.nextInt(32), timeAd));
	}

	public void getRequest() {
		int currentP = 0;
		current = null;
		if (queues.get(0).size() != 0) {
			current = queues.get(0).get(0);
			currentP = current.getPriority();
		}
		currentQueue = 0;
		for (int i = 1; i < 3; i++) {
			if (queues.get(i).size() != 0
					&& queues.get(i).get(0).getPriority() > currentP) {
				current = queues.get(i).get(0);
				currentQueue = i;
			}
		}
	}

	public double workOnRequest(double totalTime) {
		if (current == null)
			return 0;
		if (current.getTimeForComplete() < .6) {
			// System.out.println("remove from "+currentQueue);
```

```
                queues.get(currentQueue).remove(current);
                current.setTimeRemove(totalTime);
                waitTimes.add(current.getWaitTime());
                statistics.get(current.getPriority()).add(current.getWaitTime());
                return current.getTimeForComplete();
            } else {
                current.subTimeForComplete(0.6);
                queues.get(currentQueue).remove(current);
                queues.get(currentQueue).add(current);
                current.addAttempt();
                if (current.getAttempts() == 3) {
                    queues.get(0).remove(current);
                    // System.out.println("Move to 2");
                    queues.get(1).add(current);
                }
                if (current.getAttempts() == 9) {
                    // System.out.println("Move to 3");
                    queues.get(1).remove(current);
                    queues.get(2).add(current);
                }
                return 0.6;
            }
        }

        @Override
        public String toString() {
            return "Q[0]" + queues.get(0) + "\n" + "Q[1]" + queues.get(1) + "\n"
                    + "Q[2]" + queues.get(2) + "\n";
        }

        public double getAverageWait() {
            double res = 0;
            for (Double d : waitTimes) {
                res += d;
            }
            return res / waitTimes.size();
        }

        public void printPriorityStatistics() {
            System.out.println(statistics);
            for (ArrayList<Double> list : statistics) {
                double res = 0;
                for (Double d : list) {
                    res += d;
                }
                res /= list.size();
                System.out.println(Math.round(res)+","+Math.round((res-
Math.floor(res))*1000));
            }
        }
}

package lab3;

import java.util.ArrayList;

public class Test {

        private Test() {

        }

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

```
		//double lambda = 1;   // time of new requests adding
		// Prepare queue of requests
		for (double lambda = 0; lambda < 10; lambda+=.1) {

			Processor p = new Processor();
			ArrayList<Double> requestsTime = new ArrayList<Double>();
			double lambda2 = 0.1; //time for requests complete
			double totalTime = 0;
			double waitTime = 0;
			double procTime = 0;
			for (int i = 0; i < 1000; i++) {
				requestsTime.add(Generator.generateExponential(lambda));
			}
			while(totalTime<1000)  {
				//System.out.println("reqT= " + requestsTime.get(0));
				//System.out.println("procT= " + procTime);
				//System.out.println("T= "+totalTime);

				if (requestsTime.get(0) <= procTime) {
					p.addNewRequest(Generator.generateExponential(lambda2),
totalTime);
					totalTime += requestsTime.get(0);
					procTime -= requestsTime.get(0);
					requestsTime.remove(0);

	requestsTime.add(Generator.generateExponential(lambda));
				} else {
					p.getRequest();
					procTime = p.workOnRequest(totalTime);
					if (procTime==0) {
						procTime = requestsTime.get(0);
						waitTime += procTime;
					} else {
						totalTime += procTime;
						requestsTime.set(0, requestsTime.get(0) -
procTime);
						procTime = p.workOnRequest(totalTime);
					}

				}


			}
			//System.out.println(p.getWaitTimes());

	//System.out.println(Math.round(p.getAverageWait())+","+Math.round((p.getAverageW
ait()-Math.floor(p.getAverageWait()))*1000));

	//System.out.println(Math.round(waitTime/totalTime*10000)+","+Math.round((waitTim
e/totalTime*10000-Math.floor(waitTime/totalTime*10000))*1000));
			//System.out.println(waitTime/totalTime*10000);
			p.printPriorityStatistics();
		}


	}
}
```