Rešenja prvog kolokvijuma iz Operativnih sistema 2
Novembar 2016.

1. (10 poena)

a)(7) MP, LP, LP, MP, HP, MP, LP, MP, LP

b)(3) $\tau = 4$

2. (10 poena)

```
monitor Computer;
export writeX, writeY, read;

var
  x, y : real;
  readyForX, readyForY, readyToRead : boolean;
  condX, condY, condRes : condition;

procedure writeX (real r);
begin
  if (not readyForX) then condX.wait;
  x:=r;
  readyForX:=false;
  if (not readyForY) then
  begin
    readyToRead:=true;
    condRes.signal;
  end;
end;

procedure writeY (real r);
begin
  if (not readyForY) then condY.wait;
  y:=r;
  readyForY:=false;
  if (not readyForX) then
  begin
    readyToRead:=true;
    condRes.signal;
  end;
end;

function read () : real;
var temp : real;
begin
  if (not readyToRead) then condRes.wait;
  temp:=x+y;
  readyToRead:=false;
  readyForX:=true;
  readyForY:=true;
  condX.signal;
  condY.signal;
  return temp;
end;


begin
  readyForX:=true; readyForY:=true; readyToRead:=false;
end;
```

3. (10 poena)

```
public class Nalog {
            private String tekuciRacun;
            private String opis;
            private Vrsta vrsta;
            private double suma;
            public enum Vrsta {UPLATA, ISPLATA}
            public Nalog(String poruka) {
                String delovi[] = poruka.split("#");
                tekuciRacun = delovi[1];
                opis = delovi[2];
                vrsta = Vrsta.valueOf(delovi[3]);
                suma = Double.parseDouble(delovi[4]);
            }
            public Nalog(String tekuciRacun, String opis, Vrsta vrsta, double suma)
{
                this.tekuciRacun = tekuciRacun;
                this.opis = opis;
                this.vrsta = vrsta;
                this.suma = suma;
            }
            public String uPoruku() {
                return "#" + tekuciRacun + "#" + opis + "#" + vrsta.toString() +
"#" + suma + "#";
            }
            public String getTekuciRacun() {
                return tekuciRacun;
            }
            public String getOpis() {
                return opis;
            }
            public Vrsta getVrsta() {
                return vrsta;
            }
            public double getSuma() {
                return suma;
            }
        }

public class Komunikacija {
            private Socket socket;
            private PrintWriter izlaz;
            private BufferedReader ulaz;
            public Komunikacija(Socket socket) throws IOException {
                this.socket = socket;
                izlaz = new PrintWriter(socket.getOutputStream(), true);
                ulaz             =             new             BufferedReader(new
InputStreamReader(socket.getInputStream()));
            }
            public void posalji(String poruka) {
                izlaz.println(poruka);
            }
            public String primi() throws IOException {
                String poruka = ulaz.readLine();
                return poruka;
            }
            public boolean kraj() {
                return socket.isClosed();
            }
            public void zatvori() throws IOException {
                izlaz.close();
                ulaz.close();
                socket.close();
            }
        }

public class Server {
```

```
    private static final int port = 6000;
    private static final int granica = 6;
    private String registarIp;
    private int registarPort;
    private Map<String, Deque<Nalog>> nalozi = new HashMap<>();
    private Map<String, Double> racuni = new HashMap<String, Double>();
    private class Nit extends Thread {
        private Socket socket;
        public Nit(Socket klijent) throws IOException {
            this.socket = klijent;
        }
        public void run() {
            try {
                Komunikacija klijent = new Komunikacija(socket);
                while (!klijent.kraj()) {
                    String poruka = klijent.primi();
                    if (poruka == null) {
                        break;
                    }
                    Nalog nalog = new Nalog(poruka);
                    if (nalog.getVrsta() == Nalog.Vrsta.UPLATA) {
                        uplata(nalog);
                        klijent.posalji("ok");
                    } else {
                        Deque<Nalog> lista = dohvatiIsplate(nalog);
                        Komunikacija registar = new Komunikacija(new
Socket(registarIp, registarPort));
                        registar.posalji(Integer.toString(lista.size()));
                        for (Nalog n : lista) {
                            registar.posalji(nalog.uPoruku());
                        }
                        String rezultat = registar.primi();
                        if ("ok".equals(rezultat)) {
                            if (isplata(nalog)) {
                                klijent.posalji("ok");
                            } else {
                                klijent.posalji("not ok");
                            }
                        } else {
                            klijent.posalji("not ok");
                        }
                        registar.zatvori();
                    }
                }
                klijent.zatvori();
            } catch (IOException e) {
                e.printStackTrace();
            }
            System.out.println("finish");
        }
    }
    private class Klijent extends Nit {
        public Klijent(Socket klijent) throws IOException {
            super(klijent);
        }
    }
    private synchronized void uplata(Nalog nalog) {
        double suma = nalog.getSuma();
        if (racuni.containsKey(nalog.getTekuciRacun())) {
            suma += racuni.get(nalog.getTekuciRacun());
        }
        racuni.put(nalog.getTekuciRacun(), suma);
    }
    private synchronized boolean isplata(Nalog nalog) {
        double stanje = 0;
        if (racuni.containsKey(nalog.getTekuciRacun())) {
            stanje = racuni.get(nalog.getTekuciRacun());
        }
        if (stanje < nalog.getSuma()) {
```

```
                return false;
            }
            racuni.put(nalog.getTekuciRacun(), stanje - nalog.getSuma());
            return true;
        }
        private synchronized Deque<Nalog> dohvatiIsplate(Nalog nalog) {
            if (!nalozi.containsKey(nalog.getTekuciRacun())) {
                nalozi.put(nalog.getTekuciRacun(), new LinkedList<Nalog>());
            }
            Deque<Nalog> lista = nalozi.get(nalog.getTekuciRacun());
            lista.addFirst(nalog);
            while (lista.size() > granica + 1) {
                lista.removeLast();
            }
            return new LinkedList<Nalog>(lista);
        }
        public Server(String registarIp, int registarPort) throws IOException {
            this.registarIp = registarIp;
            this.registarPort = registarPort;
            ServerSocket server = new ServerSocket(port);
            while (true) {
                Socket sock = server.accept();
                Thread nit = new Nit(sock);
                nit.start();
            }
        }
        public static void main(String args[]) throws IOException {
            Server server = new Server("nbs.co.rs", 11111);
        }
    }
```