Rešenja prvog kolokvijuma iz Operativnih sistema 2 Oktobar 2015.

1. (10 poena)

```
class Scheduler {
public:
  Scheduler ();
  PCB* get ();
  void put (PCB*);
private:
  static unsigned thresholds[N];
  static const int N;
  PCB* head[N];
  PCB* tail[N];
};

Scheduler::Scheduler () {
  for (int i=0; i<N; i++)
    head[i]=tail[i]=0;
}

void Scheduler::put (PCB* pcb) {
  if (pcb==0) return; // Exception!
  int pri=0;
  for (pri=0; pri<N-1; pri++)
    if (pcb->tau<=thresholds[pri]) break;
  // Put pcb in the corresponding queue:
  pcb->next = 0;
  if (tail[pri]==0)
    tail[pri] = head[pri] = pcb;
  else
    tail[pri] = tail[pri]->next = pcb;
}

PCB* Scheduler::get () {
  for (int i=0; i<N; i++)
    if (head[i]) {
       PCB* ret = head[i];
       head[i] = ret->next;
       if (head[i]==0) tail[i]=0;
       ret->next = 0;
       return ret;
    }
  return 0;
}
```

2. (10 poena)

```
class Server {
  private Data d;
  private bool readyToRead = false, readyToWrite = true;

  public synchronized void put (Data data) {
    while (!this.readyToWrite) this.wait();
    this.readyToWrite=false;
    this.d=data;
    this.readyToRead=true;
    this.notifyAll();
  }

  public synhronized Data get () {
    while (!this.readyToRead) this.wait();
    this.readyToRead=false;
    Data data=d;
    this.readyToWrite=true;
    this.notifyAll();
    return data;
  }
}
```

3. (10 poena)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Server extends Thread {
   protected static final int REQUEST_PORT = 6000;
   protected static final int WS_RESPONSE_PORT = 6001;

   protected class ResponseCollector extends Thread {
      protected Server myServer;

      public ResponseCollector(Server s) {
         myServer = s;
      }

      public void run() {
         ServerSocket ss = null;
         try {
            ss = new ServerSocket(WS_RESPONSE_PORT);
            while (true) {
               Socket s;
               BufferedReader in = null;
               try {
                  s = ss.accept();
                  in = new BufferedReader(new InputStreamReader(
                      s.getInputStream()));
                  String response = in.readLine();
                  String[] fields = response.split("#");
                  myServer.setNumOfRequests(fields[0],
                      Integer.parseInt(fields[1]));
                  in.close();
               } catch (Exception e) {
                  // obrada greske...
               } finally {
                  if (in != null)
                     in.close();
```

```
          }
        }
      } catch (Exception e) {
        // ...
      } finally {
        try {
          ss.close();
        } catch (Exception e) {
          // ...
        }
      }
    }

  }

  protected String[] workstations;
  protected int[] numOfRequests;

  public Server(String[] workstations) {
    this.workstations = workstations;
    numOfRequests = new int[workstations.length];
    for (int i = 0; i < numOfRequests.length; i++)
      numOfRequests[i] = 0;
  }

  public void start() {
    ResponseCollector rc = new ResponseCollector(this);
    rc.start();
    super.start();
  }

  public void run() {
    ServerSocket ss = null;
    try {
      ss = new ServerSocket(REQUEST_PORT);
      while (true) {
        Socket s;
        BufferedReader in = null;
        PrintWriter out = null;
        try {
          s = ss.accept();
          in = new BufferedReader(new InputStreamReader(
               s.getInputStream()));
          out = new PrintWriter(new OutputStreamWriter(
               s.getOutputStream()), true);
          String request = in.readLine();
          if (request.equals("request")) {
            String workStation = getWorkstation();
            out.println(workStation);
          }
          in.close();
          out.close();
        } catch (Exception e) {
          // obrada greske...
        } finally {
          if (in != null)
            in.close();
          if (out != null)
            out.close();
        }
      }
    } catch (Exception e) {
      // ...
    } finally {
      try {
        ss.close();
        } catch (IOException e) {
          //
        }
      }
    }

    protected synchronized String getWorkstation() {
      int minReq = numOfRequests[0], minInd = 0;
      for (int i = 1; i < numOfRequests.length; i++) {
        if (numOfRequests[i] < minReq) {
          minReq = numOfRequests[i];
          minInd = i;
        }
      }
      numOfRequests[minInd]++;
      return workstations[minInd];
    }

    protected synchronized void setNumOfRequests(String workstation, int
  nReq) {
      for (int i = 0; i < workstations.length; i++) {
        if (workstations[i].equals(workstation)) {
          numOfRequests[i] = nReq;
          break;
        }
      }
    }

  }
```