Rešenja prvog kolokvijuma iz Operativnih sistema 2 Decembar 2015.

1. (10 poena)

```
class Scheduler {
public:
  Scheduler ();
  PCB* get (int proc);
  void put (PCB*);
private:
  static const int P, N;
  PCB* head[P][N];  // Heads and tails of ready lists
  PCB* tail[P][N];  // for P processors and N priorities
  unsigned size[P][N]; // Number of processes in each ready list
};

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

void Scheduler::put (PCB* pcb) {
  if (pcb==0) return; // Exception!
  int pri = pcb->pri;
  int proc=0;  // Processor to schedule on
  unsigned minSize = size[0][pri];
  for (int p=1; p<P-1; p++)
    if (size[p][pri]<minSize) {
      proc = p;
      minSize = size[p][pri];
    }
  // Put pcb in the corresponding queue:
  pcb->next = 0;
  if (tail[proc][pri]==0)
    tail[proc][pri] = head[proc][pri] = pcb;
  else
    tail[proc][pri] = tail[proc][pri]->next = pcb;
  size[proc][pri]++;
}

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

2. (10 poena)

```
class Gate {
  private bool isOpen = true;

  public synchronized void close () {
    this.isOpen=false;
  }

  public synchronized void open () {
    this.isOpen=true;
    this.notifyAll();
  }

  public synhronized pass () {
    while (!this.isOpen) this.wait();
  }
}
```

3. (10 poena)

```
package rs.etf;

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 class RequestHandler extends Thread {

        private Socket mySocket;
        private BufferedReader in = null;
        private PrintWriter out = null;

        public RequestHandler (Socket socket) {
            this.mySocket = socket;
        }

        public void run() {
            try {
                this.in = new BufferedReader(new InputStreamReader(
                        this.mySocket.getInputStream()));
                this.out = new PrintWriter(new OutputStreamWriter(
                        this.mySocket.getOutputStream()), true);
                while (true) {
                    // Read from this.in and write to this.out...
                }
            } catch (Exception e) {
                // ...
            } finally {
                try {
                    this.in.close();
                    this.out.close();
                    this.mySocket.close();
                } catch (Exception e) {
                    // ...
                }
            }
        }

    }
```

```
protected class RequestListener extends Thread {

    private ServerSocket mySocket = null;

    public RequestListener (int port) {
        try {
            this.mySocket = new ServerSocket(port);
        } catch (Exception e) {
            // ...
        }
    }

    public void run() {
        try {
            while (true) {
                Socket s;
                try {
                    s = this.mySocket.accept();
                    new RequestHandler(s).start();
                } catch (Exception e) {
                    // ...
                }
            }
        } catch (Exception e) {
            // ...
        } finally {
            try {
                this.mySocket.close();
            } catch (Exception e) {
                // ...
            }
        }
    }

}


protected static final int REQUEST_PORT = 6000;
protected int[] ports;
protected RequestListener[] reqListeners;

public Server (int[] ports) {
    this.ports = ports;
}

public void start() {
    this.reqListeners = new RequestListener[ports.length];
    for (int i=0; i<this.reqListeners.length; i++) {
        this.reqListeners[i] = new RequestListener(this.ports[i]);
        this.reqListeners[i].start();
    }
    super.start();
}

public void run () {
    ServerSocket ss = null;
    try {
        ss = new ServerSocket(REQUEST_PORT);
        while (true) {
            Socket s = null;
            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);
                    int request = Integer.parseInt(in.readLine());
                    String channel = this.getChannel(request);
                    out.println(channel);
                }
                catch (Exception e) {
                    // ...
                } finally {
                    if (in!=null) in.close();
                    if (out!=null) out.close();
                    if (s!=null) s.close();
                }
            }
        } catch (Exception e) {
            // ...
        } finally {
            try {
                ss.close();
            } catch (IOException e) {
                // ...
            }
        }
    }

    protected String getChannel (int req) {
        if (req>=0 && req<this.ports.length)
            return Integer.toString(this.ports[req]);
        else
            return Integer.toString(this.ports[0]);  // Default listener
    }

}
```