Rešenja prvog kolokvijuma iz Operativnih sistema 2 Oktobar 2011.

1. (10 poena) MP, HP, MP, MP, LP, MP, HP

2. (10 poena)

```
monitor server;
export acquireToken, returnToken;

var numOfTokens : integer;

procedure acquireToken ();
begin
  if numOfTokens<=0 then wait();
  numOfTokens := numOfTokens - 1;
end;

procedure returnToken ();
begin
  numOfTokens := numOfTokens + 1;
  notify();
end;

begin
  numOfTokens := N;
end; (* server *)


task type client;
begin
  loop
    server.acquireToken;
    do_some_activity;
    server.returnToken;
  end;
end; (* client *)
```

3. (10 poena)

```
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class Agent {

static int bestOffer;
static String bestClientHost = null;
static int bestClientPort;
static boolean auctionOver = false;
static int badOfferCounter = 0;
static BufferedReader in;
static PrintWriter out;

public static void main(String[] args) {
 try {
  ServerSocket sock = new ServerSocket(1025);
  Socket clientSocket = sock.accept();
  in = new BufferedReader(new InputStreamReader(
    clientSocket.getInputStream()));
  StringTokenizer st = new StringTokenizer(in.readLine(), "#");
  if (!st.nextToken().equals("StartAuction")) {
```

```
    auctionOver = true;
  }else {
    bestOffer = Integer.parseInt(st.nextToken());}

  clientSocket.close();

  while (!auctionOver) {
   clientSocket = sock.accept();

   in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
   out = new PrintWriter(clientSocket.getOutputStream(), true);
   st = new StringTokenizer(in.readLine(), "#");
   String clientHost = st.nextToken();
   String clientPort = st.nextToken();
   int newOffer = Integer.parseInt(st.nextToken());

   if (newOffer > bestOffer) {
    if (bestClientHost != null) sendMsgToBestClient("BetterOfferReceived");
    bestOffer = newOffer;
    bestClientHost = clientHost;
    bestClientPort = Integer.parseInt(clientPort);
    out.println("OfferAccepted");
    badOfferCounter = 0;
   } else {
    out.println("OfferRejected");
    badOfferCounter++;
   }

   clientSocket.close();

   if (badOfferCounter == 5) {
    if (bestClientHost != null) sendMsgToBestClient("YouWon!");
    auctionOver = true;
   }
  }
 } catch (Exception e) { System.err.println(e);}
}

static void sendMsgToBestClient(String msg) throws UnknownHostException,
IOException {
 Socket clientSocket = new Socket(bestClientHost,bestClientPort);
 PrintWriter oldOut = new PrintWriter(clientSocket.getOutputStream(),true);
 oldOut.println(msg);
 clientSocket.close();
 }
}
```

4. (10 poena)

a)(5) Dokaz kontradikcijom. Pretpostavimo da može nastati mrtva blokada, što znači da postoji zatvoren krug procesa P_{i1}, P_{i2}, ..., P_{in} (n>=1) koji su međusobno blokirani. Prema uslovima algoritma, odatle bi sledilo da je: $i_1 > i_2 > \ldots > i_n > i_1$, što ne može biti, pa mrtva blokada ne može nastati.

b)(5) Prema uslovima algoritma, ako stariji proces zatraži resurs koga drži neki mlađi proces, mlađi proces se poništava i pokreće ponovo. Kada se poništeni proces ponovo pokrene, ako bi mu se dodelio novi ID koji odgovara vremenu njegovom ponovnog pokretanja, on bi bio još mlađi u sistemu, pa bi trpeo još više poništavanja, što može dovesti do njegovog izgladnjivanja. Zato mu treba dodeliti isti ID koji je imao pri prvom pokretanju. Ako bi on bio dalje ponovo poništavan, vremenom bi taj proces postajao sve stariji i konačno postao najstariji, kada više neće doživeti poništavanje, odnosno neće trpeti izgladnjivanje.