Rešenja prvog kolokvijuma iz Operativnih sistema 2 Novembar 2017.

1. (10 poena)

```
class Scheduler {
public:
  Scheduler ();
  PCB* get ();
  void put (PCB*);
  void age ();
private:
  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!
  pcb->next = 0;
  int pri=pcb->priority;
  if (tail[pri]==0)
    tail[pri] = head[pri] = pcb;
  else
    tail[pri] = tail[pri]->next = pcb;
}

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

void Scheduler::age () {
  for (int i=1; i<N; i++)
    if (head[i]) {
      if (tail[i-1])
        tail[i-1]->next = head[i];
      else
        head[i-1] = head[i];
      tail[i-1] = tail[i];
      head[i] = tail[i] = 0;
    }
}
```

2. (10 poena)

```
monitor Toggle;
export flip, flop;

  var
    toggle : boolean;
    canFlip, canFlop : condition;

procedure flip;
begin
  while not toggle do canFlip.wait;
  (* do flip *)
  toggle := false;
  canFlop.signal;
end;

procedure flop;
begin
  while toggle do canFlop.wait;
  (* do flop *)
  toggle := true;
  canFlip.signal;
end;

begin
  toggle := false;
end;
```

3. (10 poena)

```
public class Server extends Thread {
    private Set<String> filesInUse = new HashSet<String>();
    private int port;

    public Server(int port) {
        this.port = port;
    }

    public void work() {
        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(port);
        } catch (IOException e) {
            e.printStackTrace();
            return;
        }

        while (true) {
            try {
                Socket client = serverSocket.accept();
                Thread worker = new RequestHandler(filesInUse, client);
                worker.run();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }

    public static void main(String [] args)
                           throws IOException, InterruptedException {
        Server server = new Server(5555);

        server.work();
    }
}
public class RequestHandler extends Thread {
    private static final Pattern PATTERN = Pattern.compile("line#(.*)#");
    private Set<String> filesInUse;
    private Service service;

    public RequestHandler(Set<String> filesInUse, Socket socket)
                                                 throws IOException {
        this.filesInUse = filesInUse;
        this.service = new Service(socket);
    }

    public void run() {
        String line;
        try {
            line = service.receiveMessage();
            String fileName = line;
            File file = new File(fileName);

            startUsingFile(fileName);

            receiveFile(file);

            sendFile(fileName);

            line = service.receiveMessage();
            if (!"OK".equals(line)) {
                file.delete();
            }

            finishWithFile(fileName);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    private void receiveFile(File file) throws IOException {
        String line;PrintWriter fileOut = new PrintWriter(file);

        while (true) {
            line = service.receiveMessage();
            Matcher match = PATTERN.matcher(line);
            if (match.matches()) {
                fileOut.println(match.group(1));
            } else {
                break;
            }
        }
        fileOut.close();
    }

    private void sendFile(String fileName) throws IOException {
        String line;
        BufferedReader reader =
               new BufferedReader(new FileReader(fileName));
        while ((line = reader.readLine()) != null) {
            service.sendMessage(String.format("line#%s#", line));
        }
        service.sendMessage("eof##");
        reader.close();
    }

    private void startUsingFile(String fileName)
                                          throws InterruptedException {
```

```
        synchronized (filesInUse) {
            while (filesInUse.contains(fileName)) {
                wait();
            }
            filesInUse.add(fileName);
        }
    }

    private void finishWithFile(String fileName) {
        synchronized (filesInUse) {
            filesInUse.remove(fileName);
        }
    }
}
```