# Rešenja prvog kolokvijuma iz Operativnih sistema 2
# Januar 2018.

### 1. (10 poena)

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

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

void Scheduler::put (PCB* pcb, unsigned long elapsed) {
  if (pcb==0) return; // Exception!
  int proc = pcb->affinity;  // Processor to schedule on
  if (elapsed==0) {
    pcb->execTime = 0;
    unsigned long minSize = size[0];
    proc = 0;
    for (int p=1; p<P; p++)
      if (size[p]<minSize) {
        proc = p;
        minSize = size[p];
    }
    pcb->affinity = proc;
  }
  // Put pcb in proc's queue:
  pcb->execTime += elapsed;
  PCB *prev=0, *cur=head[proc];
  while (cur && cur->execTime<=pcb->execTime) {
    prev = cur; cur = cur->next;
  }
  if (prev==0) {
    pcb->next = head[proc];
    head[proc] = pcb;
  } else {
    prev->next = pcb;
    pcb->next = cur;
  }
  size[proc]++;
}

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

**2. (10 poena)**

```
class Toggle {
  public Toggle () {}

  public synchronized flip () {
    while (!toggle) wait();
    // do flip
    toggle = false;
    notifyAll();
  }

  public synchronized flop () {
    while (toggle) wait();
    // do flop
    toggle = true;
    notifyAll();
  }

  private bool toggle = true;
}
```

**3. (10 poena)**

```
public class Server {
    private int port;
    private boolean reject = false;
    private int acceptedClients = 0;
    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();
                Service service = new Service(client);
                if (!clientAdmission(service)) continue;
                Thread worker = new RequestHandler(this, service);
               worker.start();
            } catch (IOException e) {
               e.printStackTrace();
            }
        }
    }
    public synchronized boolean clientAdmission(Service service) {
        if (reject) {
            service.sendMessage("rejected");
            return false;
        }
        acceptedClients++;
        service.sendMessage("accepted");
        return true;
    }
    public synchronized void clientFinished() {
        acceptedClients--;
        if (acceptedClients > 0) {
            reject = true;
        } else {
            reject = false;
```

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

public class RequestHandler extends Thread {

    private Server server;
    private Service service;
    public RequestHandler(Server server, Service service) throws IOException
 {
        this.service = service;
        this.server = server;
    }
    public void run() {
        try {
            String line = service.receiveMessage();
            while (!line.equals("end")) {
                line = service.receiveMessage();
            }
            server.clientFinished();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```