Rešenja prvog kolokvijuma iz Operativnih sistema 2 novembar 2020.

1. (10 poena)

```
class Scheduler {
public:
  Scheduler () {}
  PCB* get ();
  void put (PCB*, bool wasBlocked);

private:
  class ProcList {
  public:
    ProcList () : head(0), tail(0) {}
    void put (PCB* p);
    PCB* get ();
  private:
    PCB *head, *tail;
  };

  ProcList ready[MaxPri+1];
};

inline void Scheduler::ProcList::put (PCB* p) {
  if (tail) tail = tail->next = p;
  else head = tail = p;
  p->next = 0;
}

inline PCB* Scheduler::ProcList::get () {
  PCB* ret = head;
  if (head) head = head->next;
  if (!head) tail = 0;
  return ret;
}

PCB* Scheduler:: put (PCB* p, bool wasBlocked) {
  if (pcb==0) return; // Exception!
  if (wasBlocked) {
    pcb->tau=(pcb->tau+pcb->time)>>1;
    pcb->time = 0;
  }
  unsigned long pri = pcb->tau/TAU;
  if (pri>MaxPri) pri = MaxPri;
  ready[pri].put(p);
}

PCB* Scheduler::get () {
  for (int i=0; i<=MaxPri; i++) {
    PCB* p = ready[i].get();
    if (p) return p;
  }
  return 0;
}
```

2. (10 poena)

```
monitor ResourcePool;
export request, release;

  const PoolSize : integer = ...;

  var
    free : array 0..PoolSize-1 of boolean;
    numOfFree : 0..PoolSize;
    resourceAvailable : condition;

function request () : 0..PoolSize-1;
  var i : integer;
begin
  if (numOfFree=0) resourceAvailable.wait;
  numOfFree:=numOfFree-1;
  for i:=0 to PoolSize-1 do
    if free[i] then
      begin
        free[i]:=false;
        return i;
      end
end;

procedure release (i : 0..PoolSize-1);
begin
  if free[i] return;
  free[i]:=true;
  numOfFree:= numOfFree+1;
  if (numOfFree=1) resourceAvailable.signal;
end;

begin
  var i : integer;
  for i:=0 to PoolSize-1 do free[i] := true;
  numOfFree := PoolSize;
end;
```

3. (10 poena)

```
public class Server {
    public static class User {
        private String userIp;
        private int userPort;

        public User(String userIp, int userPort) {
            this.userIp = userIp;
            this.userPort = userPort;
        }

        public String getUserIp() {
            return userIp;
        }

        public int getUserPort() {
            return userPort;
        }
    }

    private User user = null;

    public synchronized User addAndMatchUser(String ip, int port) {
        User ret = user;
        if (user != null) {
            user = null;
        } else {
            user = new User(ip, port);
        }
        return ret;
    }

    public void run() {
        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(5555);

            while (true) {
                Socket clientSocket = clientSocket = serverSocket.accept();

                Service clientService = new Service(clientSocket);

                new RequestHandler(clientService, this).start();
            }

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (serverSocket != null) {
                try {
                    serverSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public static void main(String args[]) {
        Server server = new Server();
```

```
        server.run();
    }
}

public class RequestHandler extends Thread {
    private final Service service;

    private final Server server;
    public RequestHandler(Service service, Server server) {
        this.server = server;
        this.service = service;
    }

    private String parseAndRespond(String msg) {
        String[] data = msg.split("#");
        Server.User user = server.addAndMatchUser(data[1],
Integer.parseInt(data[2]));
        if (user == null) {
            return "#WaitForMatch#";
        } else {
            return String.format("#Matched#%s#%d#", user.getUserIp(),
user.getUserPort());
        }
    }

    public void run() {
        try {
            // Login
            // ...
            String msg = service.receiveMessage();
            service.sendMessage(parseAndRespond(msg));

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                service.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```