Rešenja prvog kolokvijuma iz Operativnih sistema 2 novembar 2023.

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;
  };

  static const int HP, MP, LP;
  ProcList ready[3];
};

const int Scheduler::HP = 0, Scheduler::MP = 1, Scheduler::LP = 2;

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) {
  int lowerBound = max(p->defPri-PRI_MARGIN,0);
  int upperBound = min(p->defPri+PRI_MARGIN,MAX_PRI);
  if (wasBlocked && p->pri>lowerBound) p->pri--;
  if (!wasBlocked && p->pri<upperBound) p->pri++;
  int pri = LP;
  if (p->pri<MP_PRI) pri =  MP;
  if (p->pri<HP_PRI) pri =  HP;
  ready[pri].put(p);
}

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

2. (10 poena)

```
monitor ReadersWriters;
  export startRead, stopRead, startWrite, stopWrite;

  var writers : condition;
      isWriting : boolean;
      readers : condition;
      readersCount : integer;

  procedure startRead;
  begin
    if isWriting then
      readers.wait;
    readCount := readCount + 1;
  end;

  procedure stopRead;
  begin
    readCount := readCount - 1;
    if readCount=0 then
      writers.signal;
  end;

  procedure startWrite;
  begin
    if isWriting or readersCount>0 then
      writers.wait;
    isWriting := true;
  end;

  procedure stopWrite;
  begin
    isWriting := false;
    if writers.waiting()>0 then
      writers.signal;
    else
      readers.signalAll;
  end;

begin
  isWriting := false; readersCount := 0;
end;
```

3. (10 poena)

```
public class Server {
    public record User(String username, String password, String
phoneNumber) {}
    private Map<String, User> users = new HashMap<>();

    public void run() {
        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(5555);

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

                Service service = new Service(clientSocket);
                Thread t = new RequestHandler(this, service);
                t.start();

            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public synchronized boolean addUser(String username, String password,
String phoneNumber) {
        if (users.containsKey(username)) {
            return false;
        }
        users.put(username, new User(username, password, phoneNumber));
        return true;
    }

    public synchronized boolean loginUser(String username, String password)
{
        User user = users.get(username);
        if (user == null) {
            return false;
        }
        return user.password.equals(password);
    }


    public int start2FA(String username) {
        int code = generateRandomCode();
        User user;
        synchronized (this) {
            user = users.get(username);
        }
        sendSms(user.phoneNumber, code);

        return code;
    }

    private void sendSms(String phoneNumber, int code) {…}

    private int generateRandomCode(){…}
```

```
    public static void main(String[] args) {
        new Server().run();
    }
}

public class RequestHandler extends Thread {
    private final Server server;
    private final Service service;

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

    public void run() {
            try {
                while(!login()){}
                // Use server...
            } catch (IOException e) {
                e.printStackTrace();
            }

        service.close();
    }

    private boolean login() throws IOException {
        String msg = service.receiveMsg();
        if (msg == null) {
            throw new IOException("Msg null received");
        }
        String[] data = msg.split("#");

        if ("login".equals(data[1])) {
            boolean ret = server.loginUser(data[2], data[3]);
            if (!ret) {
                service.sendMsg("Fail");
            } else {
                int codeSent = server.start2FA(data[2]);
                service.sendMsg("2FA code?");
                String msgCode = service.receiveMsg();
                if (msgCode == null) {
                    throw new IOException("Msg null received");
                }
                int code = Integer.parseInt(msgCode);
                if (code == codeSent) {
                    service.sendMsg("Success");
                    return true;
                } else {
                    service.sendMsg("Fail");
                }
            }
        } else if ("register".equals(data[1])) {
            if (server.addUser(data[2], data[3], data[4])) {
                service.sendMsg("Success");
            } else {
                service.sendMsg("Fail");
            }
        }
        return false;
    }
}
```