Rešenja kolokvijuma iz
Operativnih sistema 2, avgust 2021.

1. (10 poena)
a)(5)

```
class Scheduler {
public:
  Scheduler () : cursor(0) {}
  PCB* get ();
private:
  int cursor;
};

PCB* Scheduler::get () {
  int oldC = cursor;
  do {
    if (procs[cursor].isUsed && procs[cursor].isRunnable) {
      int c = cursor;
      if (++cursor == MAX_PROCESSES) cursor = 0;
      return &procs[c];
    }
    if (++cursor == MAX_PROCESSES) cursor = 0;
  } while (cursor != oldC);
  return 0;
}
```

a)(5)

```
class Scheduler {
public:
  Scheduler () {}
  PCB* get ();
};

PCB* Scheduler::get () {
  int maxPri = -1, maxProc = -1;
  for (int i=0; i<MAX_PROCESSES; i++) {
    if (!procs[i].isUsed || !procs[i].isRunnable) continue;
    if (procs[i].pri<maxPri) maxPri = procs[i].pri, maxProc = i;
  };
  if (maxProc>=0) return &procs[maxProc];
  else return 0;
}
```

2. (10 poena) Invarijanta protokola koja sprečava mrtvu blokadu jeste ta da u svakom trenutku svi procesi koji čekaju na resurs jesu stariji (imaju manju vremensku marku) od procesa koji taj resurs drži. To se može postići uređenjem reda čekanja u opadajući redosled po vremenskoj marki, s tim da je ova implementacija takva da je prvi u redu onaj proces koji trenutno drži resurs, a iza njega su svi procesi stariji i poređani monotono.

```
int Resource::allocate () {
  if (this->queue.isEmpty()) {
    this->queue.insert(running,running->timestamp);
    return 1;
  } else
  if (this->queue.first()->timestamp>running->timestamp) {
    this->queue.insert(running,running->timestamp);
    return 0;
  } else
    return -1;
}

PCB* Resource::release () {
  if (this->queue.first()!=running) return 0; // Error
  this->queue.removeFirst();
  return this->queue.first();
}
```

3. (10 poena)

```
void pageFault (PCB* pcb, size_t page) {
  size_t frame;
  if (pcb->usedFrames < pcb->allowedFrames) {
    frame = getFrameFromPool();
    pcb->usedFrames++;
  } else {
    size_t victimPage;
    getVictimPage(pcb,&victimPage,&frame);
    setPageUnavailable(pcb,victimPage);
    savePage(pcb,victimPage,frame);
  }
  loadPage(pcb,page,frame);
  setPageAvailable(pcb,page,frame);
}
```

4. (10 poena)

```
public class Server {
    private double balance;
    private int nextId = 0;
    private Map<Integer,Double> reservations = new HashMap<Integer, Double>();

    public Server(double balance) {
        this.balance = balance;
    }

    public synchronized String reserveAmount(double value) {
        int id = ++nextId;

        if (balance < value) {
            return "FAIL";
        }

        reservations.put(id, value);
        balance -= value;

        return String.format("OK#%d", id);
    }

    public synchronized String finalAmount(int id, double value) {
        if (!reservations.containsKey(id) || reservations.get(id) < value) {
            return "FAIL";
        }

        double reserved = reservations.remove(id);
        balance += reserved - value;
```

```
        return "OK";
    }

    public void run() {
        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(5555);

            while (balance > 0) {
                Socket clientSocket = clientSocket = serverSocket.accept();

                Service clientService = new Service(clientSocket);

                RequestHandler user = new RequestHandler(clientService, this);
                user.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(1000.0);

        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;
    }

    public void run() {
        try {
          String msg = service.receiveMessage();
          String[] data = msg.split("#");

          String response;
          if (data.length == 1) {
              response = server.reserveAmount(Double.parseDouble(data[0]));
          } else {
              response = server.finalAmount(Integer.parseInt(data[0]),
Double.parseDouble(data[1]));
          }

          service.sendMessage(response);

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

```
                e.printStackTrace();
            }
        }
    }
}
```

Klasa Service je data na vežbama.