Rešenja prvog kolokvijuma iz Operativnih sistema 2, decembar 2021.

1. (10 poena)

```
class GlobalScheduler {
public:
  GlobalScheduler () : active (0) {}

  void register (Scheduler* s, unsigned priClass);
  PCB* get ();
  void put (PCB*, bool wasBlocked);

private:
  Scheduler* scheds[2][N_SCHED_CLASS] = {};
  int active;
  static const unsigned maxPri[N_SCHED_CLASS];
};

void GlobalScheduler::register (Scheduler* s, unsigned priClass) {
  scheds[0][priClass] = s;
  scheds[1][priClass] = s->clone();
}

void GlobalScheduler::put (PCB* pcb, bool wasBlocked) {
  if (pcb==0) return; // Exception!
  for (unsigned priClass = 0; priClass < N_SCHED_CLASS; priClass++)
    if (pcb->pri <= maxPri[priClass]) {
      if (wasBlocked &&
         ((priClass>0) ?
            (pcb->pri>maxPri[priClass-1]+1):(pcb->pri>0)) pcb->pri--;
      if (!wasBlocked && pcb->pri<maxPri[priClass]) pcb->pri++;
      if (wasBlocked)
        scheds[active][priClass]->put(pcb,wasBlocked);
      else
        scheds[1-active][priClass]->put(pcb,wasBlocked);
      return;
    }
  // Exception;
}

PCB* Scheduler::get () {
  for (int i=0; i<N_SCHED_CLASS; i++) {
    if (!scheds[active][i]) continue;
    PCB* p = scheds[active][i]->get();
    if (p) return p;
  }
  active = 1-active;
  for (int i=0; i<N_SCHED_CLASS; i++) {
    if (!scheds[active][i]) continue;
    PCB* p = scheds[active][i]->get();
    if (p) return p;
  }
  return 0; // No ready processes at all!
}
```

2. (10 poena)

```
class MessageQueue {
public:
  MessageQueue () : head(0), tail(0),
                    mutex(1), msgAvailable(0), spaceAvailable(0) {}

  void put (const char* message);
  void get (char* buffer);

private:
  struct Message {
    Message* next;
    char* msg;
  };

  Message *head, *tail;
  Semaphore mutex, msgAvailable, spaceAvailable;
};

void MessageQueue::put (const char* m) {
  mutex.wait();
  Message* msg = 0; char* str = 0;
  size_t sz = sizeof(m) + 1;
  for (bool done = false; !done; ) {
    msg = (Message*)malloc(sizeof(Message));
    str = malloc(sz);
    if (msg==0 || str==0) {
      free(msg); free(str);
      Semaphore::signalWait(&mutex,&spaceAvailable);
      mutex.wait();
    } else
      done = true;
  }
  strcpy(str,m);
  msg.next = 0;
  msg.msg = str;
  if (head==0) head = tail = msg;
  else {
    tail->next = msg;
    tail = msg;
  }
  if (msgAvailable.value()<0) msgAvailable.signal();
  mutex.signal();
}

void MessageQueue::get (char* buffer) {
  mutex.wait();
  if (head==0) {
    Semaphore::signalWait(&mutex,&msgAvailable);
    mutex.wait();
  }
  Message m = head;
  head = head->next;
  if (head==0) tail = 0;
  strcpy(buffer,m.msg);
  free(m.msg); free(m);
  if (spaceAvailable.value()<0) spaceAvailable.signal();
  mutex.signal();
}
```

3. (10 poena)

```
public class Server {
    private Data allData = new Data();

    public void run() {
        ServerSocket serverSocket = null;

        CounterThread thread = new CounterThread(this);
        thread.start();
        try {
            serverSocket = new ServerSocket(5555);

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

                thread.clientArrived();

                Service service = new Service(clientSocket);

                String msg = service.receiveMsg();
                update(msg);
                service.sendMsg("Ok");
                service.close();

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

    public synchronized void update(String data) {
        allData.update(data);
    }

    public synchronized void draw() {
        allData.draw();
    }

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

public class CounterThread extends Thread {
    private static final int TIMER_MS = 10000;
    private final Server server;
    private boolean clientPresent = false;

    public CounterThread(Server server) {
        this.server = server;
    }

    public void run() {
        while (true) {
            try {
               synchronized (this) {
                   clientPresent = false;
                   wait(TIMER_MS);
                   if (clientPresent) {
                       continue;
                   }
```

```
                }
            } catch (InterruptedException e) {
                continue;
            }
            server.draw();
        }
    }

    public synchronized void clientArrived() {
        clientPresent = true;
        notify();
    }
}
```