Kolokvijum iz Operativnih sistema 1
jun 2021.

1. (10 poena)

VA(32): Segment(12).Offset(20); PA(40)

```
const unsigned SEG_WIDTH = 12, OFFS_WIDTH = 20,
               MAX_SEG_SIZE = 1U<<OFFS_WIDTH;

inline void setSMTEntry (unsigned long* smt,
                         unsigned seg, unsigned limit,
                         unsigned long paddr, unsigned short rwx) {
  smt[seg] = (paddr<<(OFFS_WIDTH+3)) | (limit<<3) | rwx;
}

void initSegment (SegDesc* sd, unsigned long* smt) {
  unsigned size = sd->size;
  unsigned saddr = sd->startAddr;
  while (size>0) {
    setSMTEntry(smt,saddr>>OFFS_WIDTH,
                (size>MAX_SEG_SIZE?MAX_SEG_SIZE-1:size-1),0,sd->rwx);
  if (size < MAX_SEG_SIZE) break;
    size -= MAX_SEG_SIZE;
    saddr += MAX_SEG_SIZE;
  }
}
```

2. (10 poena)

```
void Semaphore::wait () {
  lock(lck);
  Thread* oldRunning = Thread::runningThread;
  if (--val<0)
    this->blocked.put(oldRunning);
  else
    Scheduler::put(oldRunning);
  Thread* newRunning = Thread::runningThread = Scheduler::get();
  if (oldRunning!=newRunning)
    Thread::yield(oldRunning,newRunning);
  unlock(lck);
}

void Semaphore::signal () {
  lock(lck);
  if (++val<=0)
    Scheduler::put(this->blocked.get());
  Thread* oldRunning = Thread::runningThread;
  Scheduler::put(oldRunning);
  Thread* newRunning = Thread::runningThread = Scheduler::get();
  if (oldRunning!=newRunning)
    Thread::yield(oldRunning,newRunning);
  unlock(lck);
}
```

3. (10 poena)

```
Byte* BlockIOCache::getBlock (BlkNo blk) {
  // Find the requested block in the cache and return it if present:
  int entry = hash(blk);
  for (int i=map[entry]; i!=-1; i=entries[i].next)
    if (entries[i].blkNo==blk) {
      entries[i].refCounter++;
      return entries[i].buf;
    }
  // The block is not in the cache, find a free slot to load it:
  if (freeHead==-1) evict();
  if (freeHead==-1) return 0;  // Error: cannot find space
  int free = freeHead;
  freeHead = entries[freeHead].next;
  // Load the requested block:
  entries[free].blkNo = blk;
  entries[free].refCounter = 1;
  entries[free].next = map[entry];
  map[entry] = free;
  ioRead(dev,blk,entries[free].buf);
  return entries[free].buf;
}
```

4. (10 poena)

```
#include <sys/stat.h>
#include <fcntl.h>

class IFStream {
public:
  IFStream (const char* path);
 ~IFStream () { if (fd>=0) close(fd); }

  bool eof () const { return isEOF; }
  bool err () const { return isErr; }

  char getc ();

protected:
  inline void fetch ();

private:
  static const size_t BLOCK_SIZE = ..., CHAR_SIZE = 2;
  int fd;
  bool isEOF, isErr;
  char buffer[(BLOCK_SIZE + CHAR_SIZE - 1)/CHAR_SIZE];
  ssize_t curPos, size;
};

IFStream::IFStream (const char* path) : isEOF(false), isErr(false),
                                        curPos(-1), size(-1) {
  fd = open(path,O_RDONLY);
  if (fd==-1) { isErr = isEOF = true; return; }
  fetch();
}

inline void IFStream::fetch () {
  size = read(fd,buffer,BLOCK_SIZE);
  if (size==-1) isErr = true;
  if (size<=0) isEOF = true;
  curPos = 0;
}

char IFStream::getc () {
  if (!isErr && !isEOF && curPos<size) {
    char c = buffer[curPos++];
    if (curPos>=size)
```

```
      if (size<BLOCK_SIZE) isEOF = true;
      else fetch();
    return c;
  }
  return '\-1';
}
```