Rešenja zadataka za kolokvijum iz Operativnih sistema 1 jul 2020.

1. (10 poena)

```
typedef struct ChunkDesc {
  int offset, size;
} ChunkDesc;

void dataProcessor (void* chunk) {
  ChunkDesc* cnk = (ChunkDesc*)chunk;
  int end = cnk->offset+cnk->size;
  for (int i=cnk->offset; i<end; i++)
      processData(&data[i]);
  delete cnk;
}

void parallelProcessing (int n) {
   int chunkSz = N/n;
   int offset = 0;
   for (int i=0; i<n; i++) {
     int myChunkSz = chunkSz;
     if (i<N%n) myChunkSz++;
     ChunkDesc* chunk = new ChunkDesc;
     chunk->offset = offset; chunk->size = myChunkSz;
     offset+= myChunkSz;
     thread_create(dataProcessor,chunk);
   }
}
```

2. (10 poena)

```
void* vmalloc (SegDesc* head, void* addr, size_t size) {
  size_t sz = (size+PAGE_SIZE-1)>>PAGE_OFFS_SZ;
  SegDesc* sg = head;
  if (!addr) {
    for (; sg && sg->next; sg=sg->next) {
      size_t pg = sg->pg+sg->sz;
      if (sg->next->pg - pg >= sz) {
        if (insert_seg_desc(sg,pg,sz)<0) return 0;
        return (void*)(pg<<PAGE_OFFS_SZ);
      }
    }
    return 0;
  }
  else
  {
    size_t pg = addr>>PAGE_OFFS_SZ;
    for (; sg && sg->next; sg=sg->next) {
      if (sg->pg+sg->sz<=pg && pg+sz<=sg->next->pg) {
        if (insert_seg_desc(sg,pg,sz)<0) return 0;
        return (void*)(pg<<PAGE_OFFS_SZ);
      }
    }
    return 0;
  }
}
```

3.

```
class DoubleBuffer {
public:
  DoubleBuffer (size_t size, size_t chunkSizeProd, size_t chunkSizeCons);
  void put (const char* buffer);
  void get (char* buffer);

private:
  Semaphore inputBufReady, outputBufReady;
  char* buffer[2];
  size_t size, chunkP, chunkC, head, tail, slots, items;
  int inputBuf, outputBuf;
};

DoubleBuffer::DoubleBuffer (size_t sz, size_t cp, size_t cc)
  : inputBufReady(1), outputBufReady(0) {
  buffer[0] = new char[sz];
  buffer[1] = new char[sz];
  size = sz;
  chunkP = ((cp>0)?cp:1);
  chunkC = ((cc>0)?cc:1);
  head = tail = 0;
  slots = size; items = 0;
  inputBuf = 0; outputBuf = 1;
}

void DoubleBuffer::put (const char* buf) {
  if (slots==0) {
    inputBufReady.wait();
    outputBuf = !outputBuf;
    slots = size;
    tail = 0;
  }
  for (size_t i=0; i<chunkP; i++) {
    buffer[outputBuf][tail++] = buf[i++];
    slots--;
  }
  if (slots==0)
    outputBufReady.signal();
}

void DoubleBuffer::get (char* buf) {
  if (items==0) {
    outputBufReady.wait();
    inputBuf = !inputBuf;
    items = size;
    head = 0;
  }
  for (size_t i=0; i<chunkC; i++) {
    buf[i++]= buffer[inputBuf][head++];
    items--;
  }
  if (items==0)
    inputBufReady.signal();
}
```

4. (10 poena)

```
const char* fname = "../buffer.bin";
const unsigned int SLEEP_TIME = 5;

int send (const void* buffer, size_t size) {
  int fd;
  while (1) {
    fd = open(fname,O_WRONLY|O_CREAT|O_EXCL,S_IRUSR|S_IWUSR);
    if (fd>=0) break;
    if (errno!=EEXIST) return -1;
    sleep(SLEEP_TIME);
  }
  ret = write(fd,buffer,size) < 0 ? -1:0;
  ret |= close(fd);
  return ret;
}
```