Rešenja trećeg kolokvijuma iz Operativnih sistema 2, septembar 2015.

1. (10 poena)

```
int readBlock (unsigned blk, void* buffer) {
  static const int dsks = getNumOfDisks(), blks = getNumOfBlocks();
  d = blk%dsks;
  b = blk/dsks;
  if (b>=blks) return -1;  // Overflow
  int diskNo = disks[d].getLastRead();
  if (disks[d].isOK(1-diskNo))
    diskNo = 1-diskNo;
  if (disks[d].isOK(diskNo))
    disks[d].putReadRequest(diskNo,b,buffer));
  else
    return -2;  // Complete disk[d] pair failure

  return 0;
}
```

2. (10 poena)

```
#!/bin/bash

dir=$1

if [ $# -ne 1 ]; then
    echo "Greska: Neispravan broj argumenata."
    exit 1
fi

if ! [ -d $dir ]; then
    echo "Greska: Argument nije direktorijum."
    exit 1
fi

tmp='/tmp/os2_sep_2015.cpp'
for i in $(find $dir -name "*.cpp"); do
    cat $i | sed 's://.*$::' > $tmp
    cp $tmp $i
done;
```

3. (10 poena)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <unistd.h>

#define N ...
#define keyMem ...
#define keySem ...
#define SIZE ...
#define NUM ...
#define COUNT_MSG (NUM * N)

union semun {
  int              val;    /* Value for SETVAL */
  struct semid_ds *buf;    /* Buffer for IPC_STAT, IPC_SET */
  unsigned short  *array;  /* Array for GETALL, SETALL */
  struct seminfo  *__buf;  /* Buffer for IPC_INFO (Linux-specific) */
};

typedef unsigned Id;
struct Msg {
  Id receiver;
  . . .
};

void getMsg(struct Msg *msg, Id id);

struct Buffer {
  int head, tail, count;
  struct Msg data[SIZE];
  int sent_msg;
};

static void init_sem(int sem_id, int value)
{
  union semun arg;
  arg.val = value;
  semctl(sem_id, 0, SETVAL, arg);
}

static void sem_wait(int sem_id)
{
  struct sembuf sem_b;
  sem_b.sem_num = 0;
  sem_b.sem_op = -1;
  sem_b.sem_flg = SEM_UNDO;
  semop(sem_id, &sem_b, 1);
}

static void sem_signal(int sem_id)
{
  struct sembuf sem_b;
  sem_b.sem_num = 0;
  sem_b.sem_op = 1;
  sem_b.sem_flg = SEM_UNDO;
  semop(sem_id, &sem_b, 1);
}

static void inc(int *pvalue)
{
  *pvalue = (*pvalue + 1) % SIZE;
```

```
}

static void read_msg(struct Buffer *buffer, Id process_id, int sem_id)
{
  struct Msg msg;
  int head;
  sem_wait(sem_id);
  while (1) {
    head = buffer->head;
    if (buffer->count > 0 && buffer->data[head].receiver == process_id) {
      msg = buffer->data[head];
      inc(&buffer->head);
      buffer->count--;
      buffer->sent_msg--;
      // Save msg somewhere
    } else break;
  }
  sem_signal(sem_id);
}

int main()
{
  int shm_buffer_size;
  Id process_id = 0;
  shm_buffer_size = sizeof(struct Buffer);
  int  shmid  =  shmget(keyMem,  shm_buffer_size,  IPC_CREAT  |    S_IRUSR  |
S_IWUSR);
  struct Buffer *buffer = (struct Buffer*)shmat(shmid, 0, 0);
  int sem_id = semget(keySem, 0, 066 | IPC_CREAT);
  buffer->head = 0;
  buffer->tail = 0;
  buffer->count = 0;
  buffer->sent_msg = COUNT_MSG;
  init_sem(sem_id, 0);
  for (int i = 1; i < N; i++)
  {
    if (fork() != 0){
      process_id = i;
      break;
    }
  }
  int i = 0;
  while (i < NUM) {
    struct Msg msg;
    sem_wait(sem_id);
    if (buffer->count < SIZE) {
      getMsg(&msg, process_id);
      buffer->data[buffer->tail] = msg;
      buffer->count++;
      inc(&buffer->tail);
      i++;
    }
    sem_signal(sem_id);
    read_msg(buffer, process_id, sem_id);
  }
  while (buffer->sent_msg > 0) {
    read_msg(buffer, process_id, sem_id);
  }
  shmdt(buffer);
  shmctl(shmid, IPC_RMID, NULL);
  return 0;
}
```