# Rešenja trećeg kolokvijuma iz Operativnih sistema 2, Septembar 2012.

**1. (10 poena)**

a)(5) Jeste. Statički podaci (u ovom slučaju `vm` i `ptr`) se alociraju u prevedenom binarnom fajlu, a on se opet učitava u sopstveni virtuelni adresni prostor svakog procesa. Zbog toga će svaki proces imati svoje instance ovih podataka koji se koriste u bibliotečnoj funkciji, pa time prenose i u sistemski poziv. Naravno, sistemski pozivi (ovde realizovani prekidima) su inherentno međusobno isključivi. Tako procesi neće imati konflikte na ovim strukturama i svaki će vršiti sistemski poziv sa svojim parametrima, bez obzira na konkurentnost.

b)(5) Nije. Statički podaci se alociraju kako je gore opisano, pa niti unutar istog procesa dele iste instance ovih podataka. Kako nije obezbeđeno međusobno isključenje koda ove funkcije koja je kritična sekcija, može se dogoditi da jedna nit uđe u bibliotečnu funkciju i započne je postavivši vrednosti za `vm` i `ptr`, potom procesor preotme druga nit koja takođe uđe u ovu funkciju i postavi svoje vrednosti, prepisavši one prethodne. Na taj način nastaje konflikt i prva nit će izvršiti sistemski poziv sa pogrešnim parametrima. Da bi se ovaj problem rešio, potrebno je obezbediti međusobno isključenje u ovoj funkciji.

**2. (10 poena)**

```
#!/bin/bash
if test ! -d $1
then
  echo "Direktorijum $1 ne postoji"
  exit 1
fi
ls -l -a $1 / | grep ^- | wc -l
```

**3. (10 poena)**

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <string.h>

#define MESSAGE_Q_KEY 1
#define OP_AUCTION 111

struct requestMsg {
 long mtype; // type - id, (id==OP_AUCTION -> open auction)
 char msg[1]; // msq content: price
};

struct responseMsg {
 long mtype; // type - id
 char msg[128]; // msq content: response
};

size_t responseMsgLen = sizeof (struct responseMsg) - sizeof (long int);
size_t requestMsqLen = sizeof(char);

void sendMsgToBidder(long id, char* msg) {
 int responseMsgQueueId = msgget(MESSAGE_Q_KEY + 1, IPC_CREAT | 0666);
 struct responseMsg res_msg_buf;
 strcpy(res_msg_buf.msg, msg);
```

```
 res_msg_buf.mtype=id;
 msgsnd(responseMsgQueueId, &res_msg_buf, responseMsgLen, 0);
}

int main() {
 char bestOffer=0;
 long bestClientId = 0;
 int auctionOver = 0;
 int badOfferCounter = 0;
 int requestMsgQueueId= msgget(MESSAGE_Q_KEY, IPC_CREAT | 0666);
 struct requestMsg req_msg_buf;

 // opening auction - mtype = OP_AUCTION
 while(1){
  msgrcv(requestMsgQueueId, &req_msg_buf, requestMsqLen, 0, 0);
  if (!(req_msg_buf.mtype == OP_AUCTION))
   sendMsgToBidder(req_msg_buf.mtype,"NotActiveAuction");
  else
   break;
 }
 bestOffer = req_msg_buf.msg[0];

 while (!auctionOver){
  msgrcv(requestMsgQueueId, &req_msg_buf, requestMsqLen, 0, 0);
  int newOffer = req_msg_buf.msg[0];

  if (newOffer > bestOffer) {
      if (bestClientId != 0){
       sendMsgToBidder(bestClientId,"BetterOfferReceived"); }
      bestOffer = newOffer;
      bestClientId = req_msg_buf.mtype;
      sendMsgToBidder(req_msg_buf.mtype, "OfferAccepted");
      badOfferCounter = 0;
     } else {
   sendMsgToBidder(req_msg_buf.mtype, "OfferRejected");
   badOfferCounter++; }

     if (badOfferCounter == 5) {
      if (bestClientId != 0) sendMsgToBidder(bestClientId,"YouWon!");
      auctionOver = 1;
     }
 }
 return 0;
}
```