

## 6. Praktika

### **Multiprogramazioa: bezero/zerbitzari eredua**

#### **Kontzeptuak**

Bezero/zerbitzari eredua. Programazio eredu desberdinak maila abstraktuan eta ariketa zehatzekin.

#### **Urratsak**

- 1 Bezero/zerbitzari ereduaren funtsa.
- 2 Programazio-eredu desberdinak konkurrentzia eta sinkronizazioaren arabera.
- 3 Ariketen proba.

#### **Dokumentazioa**

- Klaseko gardenkiak. Liburuaren 5. kapitulua.
- Ariketen enuntziatua eta dokumentazioa.
- UNIX-eko laguntza: `man -s2 sistema-deia`

## Bezero/zerbitzari eredia: Adibideak

1.- **B** baliabide bati dagozkion eskaerak tratatzen dituen **zerbitzari1** izeneko programa bat eraiki behar da. Eskaerak MBX\_ZERB izeneko buzoi batean kokatzen dira, eta bezeroaren kodeak ondoko eskema jarraituko du:

```
struct eskaera {
    int bezeroa;
    char bez_buzoia[20];
    char zerbitzatzeko[40];
}
```

```
struct erantzuna {
    int bezeroa;
    char emaitza[50];
}
```

```
main(int argc, char *argv[])
{
    struct eskaera esk;
    struct erantzuna eran;
    int fd_zerb, fd_bez;
    ...
    fd_zerb = open("MBX_ZERB", O_WRONLY);
    sortu_izena(eskaera.bez_buzoia); //bezeroaren buzoiari izen bat eman
    eskaera.bezeroa = 76154221; //bezeroaren identifikadorea, kodea
    strcpy(eskaera.zerbitzatzeko, "B baliabidean 2006 errenta kalkulatu");
    mkfifo(eskaera.bez_buzoia, 0666);
    fd_bez = open(eskaera.bez_buzoia, O_RDWR);
    write(fd_zerb, &eskaera, sizeof(struct eskaera));
    read(fd_bez, &eran, sizeof(struct erantzuna));
    ...
}
```

Bi bezero ezin dute aldiberean erabili **B** baliabidea. Bezero batek eskaera sinkrono bat egiten duenean blokeatuta geratuko da bere *eskaera zerbitzatzeko den arte*. Bidaltzen den mezuan bezeroak sinkronizatzeko erabiltzen duen buzoiaren izena ere barneratzen da.

Eskaera bat zerbitzatzeko ondoko **funtzio** hau erabil daiteke:

```
zerbitzatu_B_baliabidea(char *zerbitzua)
```

2.- Aurreko zerbitzariaren **zerbitzari2** izeneko bertsio berri bat eraiki behar da, bezeroa desblokea dadin bere *eskaera zerbitzatzeko hasten denean*.

1.-A) **zerbitzari1** zerbitzariaren bertsio berri bat eraiki behar da, zerbitzatu\_B\_baliabidea, **programa** bat bada, eta programa honek, emaitza itzuliko du itzulera-kodea bezala.

```
zerbitzatu_B_baliabidea zerbitzua
```

2.-A) **zerbitzari2** zerbitzariaren bertsio berri bat eraiki behar da, zerbitzatu\_B\_baliabidea, **programa** bat bada, eta programa honek, emaitza itzuliko du itzulera-kodea bezala.

```
zerbitzatu_B_baliabidea zerbitzua
```

**3.- zerbitzari3** izeneko zerbitzari berri bat idatzi behar da. Kasu honetan **N** baliabide ditugu erabiltzaileek modu konkurrentean erabiltzeko prest. Bezeroek baliabide bat baino gehiago eska dezakete eskaera bakoitzean, baina eskaera zerbitzatuko da soilik eskatu bezain beste baliabide libre daudenean. Zerbitzatzeko ***baliabide nahikoak daudela ziurtatzen denean desblokeatuko da bezeroa***. Bezeroaren kodean aldaketa bakarra hau izango da:

```
struct eskaera {
    int bezeroa;
    char bez_buzoia[20];
    char zerbitzatzeko[40];
    int kopurua;
}
...
eskaera.kopurua = 7; //Zenbat baliabide eskatu behar dituen
...
```

Baliabidea erabiltzeko, zerbitzatu\_B\_programa izena duen **programa** erabiliko dugu:

```
zerbitzatu_B_programa zerbitzua kopurua bez_buzoia bez_kodea
```

Programa honek 4 argumentu ditu, zerbitzatzeko eskatu nahi dena, eskatutako B baliabide kopurua, bezeroaren buzoia eta bezeroaren kodea; eta programa honek, emaitza gisa —itzulera-kode moduan— erabilitako baliabide-kopurua itzuliko du. Bezeroa desblokeatu beharko da bere ***eskaera zerbitzatzeko hasten denean***.

**3.-A)** Aurreko zerbitzariaren bertsio berri bat inplementatu, baina oraingoan bezeroa desblokea dadin, zerbitzariak bere ***eskaera jasotzen duen unean***, ez azkarrago ezta beranduago ere.

**3.-B) zerbitzari3** zerbitzariaren bertsio berri bat inplementatu, oraingo honetan zerbitzatu\_B\_baliabidea **funtzio** bat bada, eta funtzio honen exekuzioak ***CPU karga handia*** dauka (*denbora asko exekutatzeko*):

```
zerbitzatu_B_baliabidea(char *zerbitzu, int kop, char *bez_buz, char *bez_kod)
```

**3.-C)** Aurreko zerbitzariaren bertsio berri bat inplementatu, kontutan izanda zerbitzatu\_B\_baliabidea funtzio bat dela (exekuzio denbora asko behar duena), eta bezeroa desblokeatu beharko dela baliabidea ***zerbitzatzeko amaitzen denean***.

**3.-D)** Aurreko zerbitzariaren bertsio berri bat inplementatu, hau da, bezero desblokeatu behar da baliabidea ***zerbitzatzeko amaitzen denean***, baina zerbitzatu\_B\_programa **programa** erabiltzen bada.

## Bezero/zerbitzari eredua: Ariketak

- 1.- Unix sistemaren gainean fax zerbitzu bat eraikitzeke asmoz, ondoko ideia jarraitu nahi da: bidali beharreko dokumentua fitxategi moduan egongo da, eta edozein erabiltzailek bere terminaletik faxak bidali ahal izatea nahi da. Azken ataza hau betetzeko jada definiturik dago komando bat. Komando horrek MBX\_FAX izeneko buzoi batean eskaera bat jartzen du, eskaeraren formatua honako hau izanik:

```
struct fax_esk {
    char igorlea[80];           /* izena eta helbidea */
    char jasotzailea[80];     /* izena eta helbidea */
    char helburu_faxa[12];    /* fax zenbakia */
    char fitxategia[20];     /* bidali beharreko testua */
}
```

Fax bat linea jakin batetik bidaltzeko programa berezi bat dago —honen programaziorako hainbat S/Iko portu erabili behar da— `bidali_fax` izeneko.

```
bidali_fax linea igorlea jasotzailea helburu_faxa fitxategia
```

Programa honek linea —argumentu bezala jaso duen parametroa— itzultzen du itzulera-kodean, libratu den linearen berri emateko asmoz.

Programa ezazu fax linea guztien kudeaketa egingo duen zerbitzaria ondoko bi kasuetan:

- A) Fax-linea bakarra dagoela suposatuz.
- B) Faxak bidaltzeko linea kopuru finko maximo bat dago, `N_LINEAK`, eta fax bat edozein linea batetik bidal daiteke, erabiltzen den linea zenbakia erabiltzaileari bost axola zaiolarik. Momenturen batean linea librerik ez badago mezuaren bidalketa atzeratuko da, baina, erabiltzailea modu asinkronoan luzatzen duenez bere eskaera ez da sinkronizazio arazorik izango.

- 2.- Telebistan modan dagoen “Arreba Txikia” Reality Show-aren azkeneko astean sartu gara, eta nominazioak argitzeko bozketa sistema bat ezarri nahi da. Ikusleek SMS bidez bozka dezakete, eta bozketa posible egiteko Reality\_Zerbitzaria martxan ipiniko da, bozketak jaso eta irabazlea zein den kalkulatzeko. Horretarako datu-base bat erabiliko da.

Bezeroek (ikusleek) eskaera bat bidaliko dute SMS bat bidaliz mugikorraren bitartez, eta eskaera MBX\_REALITY buzoian idatziko da, hurrengo formatua duelarik:

```
struct bozketa {
    char zenbakia[12];      /* mugikor zenbakia */
    char sms[80];         /* mezua */
}
```

Bezeroaren programa ***asinkronoa*** da, hau da, eskaera bidali eta amaituko da, erantzunaren zain geratu gabe.

Ikusle bakoitzak behin bakarrik bozka dezake. Ikusleek bozketan tranparik egin ez dezaten aztertu\_bozketa ***programa*** dugu bozka bakoitza aztertzeko. Programa honek bi argumentu ditu, ikuslearen mugikor zenbakia, eta bidalitako SMS mezua. Ikusleak aurretik ea bozkatu duen edo ez aztertzen du, eta bozkatu badu baztertu egiten da bozka. Ez badu bozkatu, aldiz, bozka berria datu-basean gehitzen da eta bozka kontuan hartzen da.

```
aztertu_bozketa zenbakia sms
```

Programa honek datu-basea atzitzen duenez, eta datu-baseetan atzipen kopurua mugatuta dagoenez, gehienez 1000 atzipen egin daitezke uneoro modu konkurrentean aztertu\_bozketa programaren bitartez.

Hurrengo eskatzen da:

- a) Marraz ezazu eskema bat sistema osatuko duten prozesu guztiak azalduz eta elkarren arteko komunikazio mekanismoak argi adieraziz.
- b) Idatz ezazu zerbitzariaren eta lagungarri ikusten dituzun beste prozesu guztien kodea.
- c) Adierazi zer aldatu beharko litzatekeen, bezero bakoitzari bere eskaera guztiz zerbitzatu ondoren abisatu beharko balitzaio. Azaldu aldaketak eta marraztu eskema berria. Ez da eredu berria kodetu behar, soilik marraztu eta azaldu.

3.- Lurralde bateko mugetan kontrol-mekanismo bat ezarri nahi da kontrolatzeko nor sartzen den eta nor ateratzen den. Horretarako, lurraldearen muga bakoitzean makina berezi bat ezarri da, zeinak hatz-marka digitalak hartu eta muga zeharkatu behar duen pertsona ea terrorista den edo ez aztertuko duen. Funtzionamendua hurrengoa litzateke: mugan dauden makinak GENERATRIX izeneko UNIX zerbitzari batera konektatuta daude, eta norbait muga zeharkatu behar duen bakoitzean, mugako makinak eskaera bat bidaltzen dio zerbitzariari, azken honek aztertuko duelarik ea terrorista den edo ez muga zeharkatu nahi duen pertsona. Mugan dauden makinak soilik eskaera bat egin ahalko dute uneko, zeinak bezero prozesu bat sortuko duen eta honek eskaera MAILTRIX izena duen zerbitzariaren buzoian idatziko duen, eskaeraren formatua hurrengoa izanik:

```
struct eskaera {
char pasaporte[20]; /* pasaporte-zenbakia */
char hatz_marka[1024]; /* hatz-marka digitala */
char buzoia[80]; /* bezeroaren buzoia */
}
```

Aztertzeko ea muga zeharkatu behar duen pertsona terrorista den edo ez, hurrengo **funtzioa** erabiliko dugu:

```
int hatz_marka_aztertu(char *hatz_marka);
```

Funtzio honek parametro bezala pasatako hatz\_marka aztertu, eta muga zeharkatu behar duen pertsona terrorista bada 1 itzultzen du, eta 0 bestela. Guztiarekin eskatzen da, hatz\_marka\_aztertu funtzioaren emaitzaren arabera bezeroari erantzutea, mezuak hurrengo formatua izango duelarik:

```
struct erantzuna {
int kodea; /* 1: terrorista; 0: ez terrorista */
}
```

GENERATRIX zerbitzariak gehienez **100** eskaera zerbitzatu ditzake uneoro modu konkurrentean, hatz\_marka\_aztertu funtzioak **CPU kontsumo altua** baitu.

Hurrengoa eskatzen da:

- a) Marraz ezazu eskema bat sistema osatuko duten prozesu guztiak azalduz eta elkarren arteko komunikazio mekanismoak argi adieraziz.
- b) Idatz ezazu bezeroaren, zerbitzariaren eta lagungarri ikusten dituzun beste prozesu guztien kodea.
- c) Adierazi zer aldatu beharko litzatekeen, hatz\_marka\_aztertu funtzioa, funtzioa izan beharrean programa bat izango balitz. Azaldu aldaketak eta marraztu eskema berria. Ez da eredu berria kodetu behar, soilik marraztu eta azaldu.

4.- Bezero/Zerbitzari sistema bat eraiki nahi da, bezeroen eskaerak (irudiak) jaso, hauek prozesatu eta emaitzak datu base batean gordetzen dituena. Irudien prozesaketa burutzeko (prozesaketa OCR – *karaktereen erazagupen optikoa* – teknikaren bidez egingo da) **funtzio** bat dugu, irudiaren fitxategiaren izena emanda, irudi prozesatua emaitza bezala bigarren parametroan itzultzen duena. Emaitzaren luzera 10 karakterekoa izango da.

```
void OCR(char *irudi_fitx, char *emaitza);
```

OCR funtzioak CPU denbora asko behar duenez irudi bat prozesatzeko, **gehienez 50 irudi prozesa daitezkeela uneoro modu konkurrentean erabaki da.**

Emaitza datu basean gehitzeko hurrengo **funtzioa** dugu, eskaeraren informazioa eta emaitza parametro bezala emanda, informazio guztia datu basean txertatzen duena:

```
void DB_gehitu(int bezero_id, char *data, char *irudi_fitx,  
              char *emaitza);
```

Sistemaren funtzionamendua honakoa da: bezero bakoitzak MBX\_ZERB\_OCR izeneko buzoira prozesatu nahi duen irudiaren eskaera bidaliko du. Eskaeraren formatua hurrengoa da:

```
struct eskaera {  
    int bezero_id;          /* Bezeroaren identifikadorea */  
    char data[8];          /* Eskaeraren data */  
    char irudi_fitx[80];    /* Irudia dagoen fitxategiaren izena */  
    char bez_buzoi[80];    /* Bezeroaren buzoia */  
  
}
```

*/\* Bezeroaren programa \*/*

```
void main(int argc, char *argv[])  
{  
    int fd1, fd2;  
    struct eskaera esk;  
    char emaitza[10];  
  
    ... /* eskaera osatu */  
    mkfifo("Nire_Buzoiaxx", 0666); /* erantzun buzoia sortu */  
    fd1 = open("MBX_ZERB_OCR", O_WRONLY);  
    fd2 = open("Nire_Buzoiaxx", O_RDWR);  
    write(fd1, &esk, sizeof(struct eskaera));  
    read(fd2, emaitza, 10);  
    close(fd1); close(fd2);  
}
```

Sistemak eskaera jaso, irudia prozesatu (gehienez 50 irudi konkurrentzian) eta emaitza datu basean idatzi beharko du. Bezeroari erantzun mezu bat bidali behar zaio, bere buzoian “PROZESATZEN” testu-katea idatziz, **irudia prozesatzen hasten denean.**

- a) Marraz ezazu eskema bat sistema osatuko duten prozesu guztiak azalduz eta elkarren arteko komunikazio mekanismoak argi adieraziz.
- b) Idatz ezazu zerbitzariaren eta lagungarri ikusten dituzun beste prozesu guztien kodea.

5.- Bezero/Zerbitzari sistema bat eraiki nahi da, MP3 motako fitxategiak deskargatzeko. Sistemaren funtzionamendua honakoa da: bezero bakoitzak MBX\_ZERB\_MP3 izeneko buzoira deskargatu nahi dituen abestien eskaera bidaliko du. Eskaeraren formatua hurrengoa da:

```
struct eskaera {
    char erabiltzailea[20]; /* erabiltzailearen izena */
    int abesti_kop; /* abesti kopurua (1 eta 5 artean) */
    char abestiak[250]; /* abesti bakoitzak 50 karaktere */
    char bez_buzoia[40]; /* bezeroaren buzoia */
}
```

Zerbitzariak 100 abesti deskargatu ditzake aldi berean. Eskaerak osorik zerbitzatu behar dira, hau da, eskatutako abesti guztiak. Eskaera bat ezin bada zerbitzatu behar beste deskarga libre ez daudelako, itxaron egin beharko da.

Bezeroak eskaera bidaltzen duenean, erantzunaren zain blokeatzen da bere buzoian. Erantzuna bidali beharko zaio eskaera guztiz burututa dagoenean, “ESKAERA ZERBITZATUA” mezu baten bidez.

Bestalde, jasotako eskaera bakoitzeko, konprobatu behar da ea erabiltzailea baimendua dagoen ala ez abestiak deskargatzeko. Hau burutzeko oso azkarra den honako **funtzioa** dugu:

```
int erabiltzailea_konprobatu(char *erabiltzailea);
```

Funtzio honek 0 bueltatzen du erabiltzailea baimendua dagoenean, eta -1 baimendua ez dagoenean. Azken kasu hau gertatzen bada, bezeroari “ESKAERA BAZTERTUA” mezua bidez erantzungo zaio, eta eskaera ez da zerbitzatuko.

Azkenik, eskaera baten abesti kopurua ez bada 1 eta 5 artekoa, orduan ere “ESKAERA BAZTERTUA” mezua bidez erantzungo zaio, eta eskaera ez da zerbitzatuko.

Eskaerak zerbitzatzeko honako **programa** dugu:

```
abestiak_deskargatu kopurua abestiak
```

Programa honek argumentu bezala pasatako abestien deskarga abiatuko du. Bukatzerakoan, deskargatutako abesti kopurua itzuliko du.

- a) Marraz ezazu eskema bat sistema osatuko duten prozesu guztiak azalduz eta elkarren arteko komunikazio mekanismoak argi adieraziz.
- b) Idatz ezazu zerbitzariaren eta lagungarri ikusten dituzun beste prozesu guztien kodea (bezeroaren atala kodetuzat jotzen da, ez egin).
- c) Eskaera guztiz burutu ondoren bezeroari erantzuna bidali beharrean, orain eskaera zerbitzatzen hasten denean bidali nahi da. Adierazi ezazu zer aldaketa izango lukeen (a) ataleko eskeman, eta marraztu eskema berria.

- 6.- Zure probintziako ogasunaren informatika zerbitzuan kontratatu zaituzte, eta errenta aitortzen erabiltzen den software berria programatzeko eskatu dizute. **errenta\_online** aplikazio honen helburua errenta aitortzea Internet bitartez egin behar duten pertsonen datuen egiaztaketa egitea da, horretarako ogasunak duen datu-baseko informazioaz baliatuz.

Sistemaren funtzionamendua ondokoa izango da: pertsona bakoitzak errenta aitortzen proposamenak MBX\_OGASUN izeneko buzoia batean idatziko du, bertan idatzen den proposamen bakoitzak ondoko formatua izango duela:

```
struct proposamena {
    char ena[80];           /* nortasun agiriaren zenbakia */
    char email[80];        /* email kontua */
    char info[250];        /* proposatutako errenta
                           aitorpenaren informazioa */
}
```

Gure aplikazioak proposatutako aitortzea onartzea emango den ala ez erabaki beharko du. Horretarako OGASUNA.DB izeneko datu-base bat erabili beharko da. Datu-baseko informazioarekin lan egiteko, beste informatikari talde batek ondoko funtzioa programatu du dagoeneko:

```
int ErrentaOnartzea(char *ena, char *info, char *datubasea);
```

zeinak pertsonaren nortasun-agiri zenbakia, errenta aitortzen informazioa, eta datu-basearen izena emanik, probintziako ogasunak estimatutako errenta aitortzarekin alderatu eta aitortzea onartu ala ukatzen den (0 ala -1 aldez aurre) itzuliko duen. Funtzio honek bere eginkizuna betetzeko CPU denbora dextente behar du.

Gure sistemak aurreko funtzioak itzultzen duen emaitzaren arabera, email bat bidaliko dio pertsonari bere proposamena onartu ala ukatzen dela abisatuz. Horretarako, inplementatuta dagoen beste funtzio hau erabiliko da:

```
void BidaliEmail(char *email_kontua, char *mezua);
```

Azkenik, gure sistemak beste ezaugarri hau bete behar du ere: pertsona desberdinen proposamenak konkurrentzian aztertuko dira, baina gehienez 100 azterketa egon beharko dira martxan uneoro.

7.- Internet-era konektatzeko eskaerak kudeatuko dituen zerbitzu bat idatzi nahi da, jadanik erregistraturik dauden erabiltzaileentzat. Horretarako, zerbitzu honek N linea kudeatuko ditu (bakoitza IP helbide bati lotuta). Konektatzeko eskaera bakoitza, baldin eta linearik libre badago, jadantik idatzita dagoen honako programaren bidez zerbitzatzen da

```
konexioa_zerbitzatu bezeroa linea_zenbakia
```

Programa honek bukatzean bezeroa konektaturik egon den denbora (segundotan) bueltatzen du. Bezeroen eskaerak honako informazioaz osaturik daude: izena, gakoa, eta erantzuna jasotzeko buzoia izena. Erantzunean esleitutako linea zenbakia bidaliko zaio bezeroari. Eskaera mezuen egitura honakoa da:

```
struct eskaera_konexioa {
    char bezeroa[20];
    char gakoa[20];
    char erantzun_buzoia[80];
}
```

Eskaera bat jasotzean, zerbitzariak lehendabizi bezeroa erregistraturik dagoen ala ez egiaztatuko du, jadantik idatzitako honako funtzioaren bidez:

```
int bezeroa_egiaztatu(char *bezeroa, char *gakoa);
```

Funtzio honek 0 bueltatzen du bezeroa erregistraturik badago, eta -1 ez badago. Egiaztapena ez bada arrakastatsua, errore mezu batekin erantzungo zaio bezeroari bere buzoian. Aldiz, egiaztapena arrakastatsua bada, konexioa zerbitzatzen hasi bezain pronto bezeroari abisatuko zaio bere buzoian mezua idatziz. Bai kasu batean eta bai bestean, erantzun mezuen egitura honakoa da:

```
struct erantzuna_konexioa {
    int emaitza;          /* 0 arrakasta, -1 porrota */
    int linea_zenbakia;  /* esleitutako linea */
}
```

Azkenik, zerbitzatutako konexio guztiei buruzko informazioa (bezeroa + konexioaren iraupena segundotan) gorde nahi da fitxategi batean. Horretarako, konexio bakoitzaren bukaeran aipatutako informazioak gehituko dira fitxategira, jadantik idatzitako honako funtzioaren bidez:

```
int konexioa_erregistratu(char *bezeroa, int iraupena);
```

Honakoa eskatzen da:

- Marraz ezazu eskema bat sistema osatuko duten prozesu guztiak azalduz eta elkarren arteko komunikazio mekanismoak argi adieraziz.

Idatz ezazu zerbitzariaren eta lagungarri ikusten dituzun beste prozesu guztien kodea.

**8.-** Zure enpresak sistema informatiko berri bat eraiki behar du, eta zure lana da sistema osoaren diseinua eta programaketa. Sistema informatikoaren helburua oso inportantea da: pneumonia atipikoaren analisia egiten duten makina batzuen kudeaketa. Horretarako analisiak egiten dituzten 7 makina dauzkazu, baina makina bakoitzak analisi bakarra egin dezake une bakoitzean. Garatuko duzun sistema hau aireportu internazional batean instalatuko da, eta bertan analisia egin beharko zaie lehenbailehen bidariari askori.

Sistema diseinatzeko bezero-zerbitzari eredua erabiltzea erabaki da, bezeroek exekutatu duten funtzioan parametro bezala pasako delarik analisia egin beharreko odol-laginaren kode digitala —void bezeroa (char \*lagin\_fitxategia)— eta eskaera bat egingo da MBX\_ANALISIRAKO izeneko buzoi batean. Sistema guztiaren kudeaketa Analisisien-gestore prozesu batek egingo du zeinak eskaerak jaso eta zerbitzatu egingo dituen. Bidaiari baten laginaren analisia egiteko ondoko programa erabiliko da

```
aztertu_lagina makina_zenbakia lagin_fitxategia
=> 0 normala, 1 gaixorik
```

Bezero baten analisia bukatu bezain pronto zerbitzariak bezeroari adierazi behar dio zein izan den erantzuna bezeroaren buzoi pertsonal baten bitartez. Erantzunaren larritasuna dela eta, analisiak egiteko makina bat ere ezingo da lanik gabe mantendu eskaerarik zerbitzatzeko prest egonez gero.

Ondokoa eskatzen da:

- Marraz ezazu eskema bat sistema osatuko duten prozesuak adieraziz eta beren arteko komunikazio mekanismoak azalduz.
- Idatz ezazu bezeroaren kodea.

Idatz ezazu zerbitzariaren eta beharrezko ikusten dituzun beste prozesu guztien kodea.

**9.-** NxN neurriko matrizeen eragiketak burutzeko zerbitzu bat garatu nahi da. Zerbitzu honen bezeroek buzoi konpartitu batera bidaliko dituzte beren eskaerak, buzoi honetatik eskaerak zerbitzari batek jasoko dituelarik. Behin eskaera bat zerbitzatu izan denean (hau da, eragiketa burutu denean), bezeroari ohartaraziko zaio buzoi partikular batera “BURUTUA” mezua bidaliz. Buzoi partikular honen izena bezeroaren eskaeran etorriko da.

Garatu nahi den zerbitzuak matrizeen batuketak, kenketak eta biderketak burutzeko gai izango da. Dagoeneko badauzkagu programatuta honako funtzioak:

```
void matrizeen_batuketa(char *f1, char *f2, char *f_emaizta);  
void matrizeen_kenketa(char *f1, char *f2, char *f_emaizta);  
void matrizeen_biderketa(char*f1, char *f2, char *f_emaizta);
```

Funtzio hauek parametro bezala hiru fitxategien izenak dituzte. Alde batetik, eragigai diren bi matrizeak gordetzen dituzten bi fitxategien izenak (f1 eta f2), eta bestalde emaitza matrizea idatzi beharko den fitxategiaren izena (f\_emaizta).

Bezeroen eskaeretan, buzoi partikularraren izenaz gain, eragiketari dagokion hiru fitxategien izenak joango dira, eta baita burutu nahi den eragiketaren kodea ere: batuketa (1), kenketa (2), edota biderketa (3).

Matrizeen arteko batuketak eta kenketak nahiko eragiketa azkarrak direnez, bi hauek burutzeko prozesu bakar bat erabiliko da. Aldiz, matrizeen arteko biderketak kalkulu-denbora gehiago behar dutenez, eta bezeroen eskaerak ahalik eta azkarren zerbitzatzeko asmoz, biderketa motako eragiketak zerbitzariak zuzenean ez burutzea erabaki da.

Ondokoa eskatzen da:

- Marraz ezazu eskema bat sistema osatuko duten prozesu guztiak azalduz eta elkarren arteko komunikazio mekanismoak argi adieraziz.
- Idatz ezazu bezeroaren, zerbitzariaren eta beharrezko ikusten dituzun beste prozesu guztien kodea.

Aztertu ezazu nola hobetu litekeen zerbitzuaren eraginkortasuna.