

eman ta zabal zazu



Universidad Euskal Herriko  
del País Vasco Unibertsitatea  
**INFORMATIKA FAKULTATEA**

**2008/2009 ikasturtea**

## Sistema Eragileak I

Konputagailuen Arkitektura eta Teknologia Saila  
Dpto. de Arquitectura y Tecnología de Computadores

**2009ko ekainaren 2a**

## ARIKETAK

### 1. ariketa [1 puntu]

---

1. Bedi honako scripta, **ez\_dakit** izenekoa.

```
if [ $# -ne 4 ]
then
    echo "1_mezua"
    exit 1
fi
if [ -f $2 -a -f $4 ]
then
    if [ -d $1 -a -d $3 ]
    then
        if [ -f $1/$2 -a -f $3/$2 ]
        then
            echo "2_mezua"
        fi
        if [ -f $1/$4 -a -f $3/$4 ]
        then
            echo "3_mezua"
        fi
    fi
fi
```

a) Zer egiten du **ez\_dakit** ?

b) Kodetu ezazu **iturburuak\_konpilatu** scripta, hurrengoa egin dezan: argumentu bezala katalogo bat jasoko du, eta katalogo horretan dauden C-ko iturburu fitxategi bakoitzeko (".c" amaiera dutenak), konpilatu egingo ditu, eta konpiladoreak emandako erroreak "erroreak.log" fitxategian metatu beharko ditu, zeina argumentu bezala pasatako katalogoan kokatuko den. Konpilatzeko "gcc" komandoaz baliatu zitezke, zeinak errore mezuak irteera estandarrean idazten dituen.

Konpilazio adibidea: gcc -c iturburu1.c

Erabilera adibidea:

**iturburuak\_konpilatu lab3**

## 2. ariketa [2 puntu]

---

UNIXeko Sarrera/Irteerako sistema-deiak erabiliz (ikus azken orriko zerrenda), ondoko programa idatzi nahi dugu C lengoaian:

```
sarrera_anitz katalogo_1 katalogo_2 ... katalogo_n
```

Programa honek argumentu bezala pasatako `katalogo_i` katalogo bakoitzaren barruan dauden sarreraren artean, hardware lotura bat baino gehiago dutenen izena idatzi beharko du irteera estandarrean, hurrengo formatua mantenduz:

```
katalogo_i/sarrera_izena:kopurua
```

Non, `katalogo_i` argumentu bezala pasatako katalogo bakoitza izango den, `sarrera_izena`, baldintza betetzen duen sarreraren izena izango den, eta `kopurua` sarrerak duen hardware lotura kopurua.

Programak konprobatu beharko du gutxienez argumentu bat pasatzen zaiola, eta argumentu bakoitza, benetan, katalogo bat dela. Argumentua ez bada katalogoa hurrengo mezua idatziko du:

```
katalogo_i: ez da katalogoa
```

## 3. ariketa [1,5 puntu]

---

Kodetu `prozesu_kopurua` programa, zeinak irteera estandarrean, argumentu bezala pasatako erabiltzaileak UNIX makinan martxan dituen prozesu kopurua idatziko duen. Makinan exekutatzen dauden prozesu guztien zerrenda lortzeko "**ps -aux**" komandoa exekutatu behar da, zeinak irteera estandarrean prozesu bakoitzaren informazioa idazten duen lerro bakoitzeko; informazio honetan prozesua sortu duen erabiltzailearen izena agertzen da, besteak beste.

Programa honen bi bertsio egitea eskatzen zaizu:

- Lehen bertsio bat UNIXeko ***komando lerro bakar bat*** erabiliz (suposatu erabiltzailea ***acaf0000*** dela).
- Bigarren bertsio ***multiprogramatu*** bat C lengoaiatz, UNIXeko ***komandoak*** erabiliz (ikus bukaerako zerrenda) eta beharrezko iruditzen zaizkizun izenik gabeko buzoiez (*pipe*-ak) baliatuz.

Erabilpen adibidea:

```
prozesu_kopurua acaf0149
```

## 4. ariketa [2,5 puntu]

DIF\_BANK banketxeak bezeroentzako kontsulta zerbitzu berri bat jarri nahi du, Bezero/Zerbitzari eredian oinarrituta, eta zerbitzu hau UNIX makina batean martxan ipiniko da. Eskaeren konfidentzialtasuna ziurtatzeko, informazioa zifratuta bidaltzen da. Beraz, eskaerak zifratuta zerbitzariaren **MBX\_DIF\_BANK** buzoira bidaliko dira, **struct eskaera** egituraren oinarrituta. Bezeroa zerbitzariaren erantzunaren zain geratuko da, **struct erantzuna** egitura izango duena.

```
struct eskaera {
    int bez_ID;
    char bez_buzoia[120];
    char kontsulta_zifratua[1024];
}
```

```
struct erantzuna {
    int bez_ID;
    char emaitza_zifratua[1024];
}
```

Bezeroek bidaltzen duten eskaera zifratua iristen zaio zerbitzariari, eta eskaera deszifratu behar da, ondoren kontsulta datu-basean egin ahal izateko. Behin kontsulta egin denean, bezeroari erantzuna zifratuta bidaliko zaio. Eragiketa guzti hauek burutzeko hurrengo **funtzioak** ditugu:

- void **deszifratu**(char \*kontsulta\_zifratua, char \*DB\_kontsulta);  
Funtzio honek kontsulta zifratua jasoko du, eta bigarren argumentuan kontsulta deszifratua itzuliko du. Funtzio honen exekuzioa oso azkarra da.
- void **zifratu**(char \*DB\_emaitza, int bez\_ID, char \*emaitza\_zifratua);  
Datu-basean kontsulta egin ondoren, emaitza bezala jaso duguna, eta bezeroaren identifikadorea jasota, hirugarren parametroan kontsultaren emaitza zifratua itzuliko du. Funtzio honen exekuzioa ere oso azkarra da.
- void **DBkontsulta**(char \*DB\_kontsulta, char \*DB\_emaitza);  
Lehen parametro bezala pasatako kontsulta datu-basean burutzen du, eta bigarren parametroan emaitza itzultzen du. Funtzio honen exekuzioak denbora asko behar du, datu-base bat atzitzea behartua baitago, disko atzipen ugari burutuz.

Zerbitzaria ahalik eta eskaera gehien zerbitzatzeko saiatu behar da, baina datu-basearen atzipenean arazoak ekiditearren, gehienez 100 eskaera zerbitzatu daitezke konkurrentzian.

Hurrengo eskatzen da:

1. Marraz ezazu eskema bat sistema osatuko duten prozesu guztiak azalduz eta elkarren arteko komunikazio mekanismoak argi adieraziz.
2. Idatz ezazu zerbitzariaren eta lagungarri ikusten dituzun beste prozesu guztien kodea.
3. Demagun bezeroaren eskaera datu-basean kontsultatzeko funtzio bat izan beharrean, hurrengo programa dugula:

```
DBquery DB_kontsulta bez_ID bez_buzoia
```

Programa honek, argumentu bezala, burutu nahi den kontsulta, bezeroaren identifikadorea eta bezeroaren buzoia jasoko ditu. Lehen argumentuarekin burutu behar den kontsulta aurrera eramango du datu-basea atzitzuz, eta emaitza duenean, erantzuna eraikiko du eta erantzuna bezeroaren buzoira bidaliko du, hirugarren argumentuaren bitartez. Marraztu ezazu eskema berri bat azalduz zein aldaketa burutuko zenukeen aurreko bertsioarekin alderatuz (ez da ezer kodetu behar, soilik aldaketak argi azaltzea).

**Sistema-deiak:**

```

int open(char *izena, int modua, int baimenak);
int close(int kanala);
int read(int kanala, char *buf, int luz);
int write(int kanala, char *buf, int luz);
int lseek(int kanala, long displ, int nondik);
    //nondik: 0 hasiera, 1 unekoa, 2 bukaera
int link(char *iturburua, char *helburua);
int unlink(char *izena);
int stat(char *izena, struct stat *sbuf);
int fstat(int kanala, struct stat *sbuf);
int chdir(char *izena);
int fork();
int execlp(char *izena, char *arg0, char *arg1, ..., char *argn, NULL);
int execvp(char *izena, char *arg[]);
int wait(int *egoera);
void exit(int egoera);
unsigned long time(0);
int pipe(int *pfd);
int mkfifo(char *izena, int baimenak);
int mknod(char *izena, int baimenak, int unitatea);
int dup(int kanala);
DIR *opendir(char *path);
int closedir(DIR *dir);
struct dirent *readdir(DIR *dir);
    struct dirent { long d_ino;        // Inode zenbakia
                    char *d_name; } // Izena
    struct stat {   st_ino:    inodo zenbakia
                    st_mode:  modua
                    st_nlink: lotura kopurua
                    st_uid:   jabearen identifikatzailea
                    st_gid:   erabiltzaile-taldearen identifikatzailea
                    st_size:  tamaina bytetan
                    st_atime: azken atzipenaren data
                    st_mtime: azken aldaketaren data
                    st_ctime: sortu zeneko data };

```

POSIX makroak fitxategiaren mota konprobatzeko (**m struct stat** aldagai baten **st\_mode** eremua da):  
 S\_ISLNK(m), S\_ISREG(m), S\_ISDIR(m), S\_ISFIFO(m)

**Funtzioak:**

```

char *strcpy(char *helburu, char *jatorri);
char *strcat(char *katea1, char *katea2);
int strcmp(char *katea1, char *katea2);
int strlen(char *katea);
char *strstr(char *katea, char *azpikatea);
int atoi(char *katea);
char *itoa(int zenbakia);
int sprintf(char *katea, char *formatua, ...);

```

**UNIX-eko programa eta utilitateak:**

```

chmod - fitxategien baimenak aldatzeko
echo - mezuak idazten ditu irteera estandarrean
grep - azpikateak bilatzen ditu fitxategietan
head - fitxategien lehen lerroak aukeratzen ditu
last - sistemara egindako konexioak pantailaratzen ditu
ps - prozesuei buruzko informazioa
sort - fitxategien lerroak alfabetikoki sailkatzen ditu
tail - fitxategien azken lerroak aukeratzen ditu
wc - lerroak, hitzak, eta karaktereak kontatzen ditu
who - sistemara konektatutako erabiltzaileei buruzko informazioa

```