

eman ta zabal zazu



Universidad Euskal Herriko
del País Vasco Unibertsitatea
INFORMATIKA FAKULTATEA

2006/2007 Ikasturtea

Sistema Eragileak I

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

2007ko maiatzaren 29a

1. ariketa [1 puntu]

a) Ondoko **auskalo** izeneko scripta emanik, azal ezazu zer egiten duen.

```
for i in $*
do
    if [ -f $i ]
    then
        echo $i
        for k in *
        do
            if [ -d $k ]
            then
                if [ -f $k/$i ]
                then
                    echo "Mezua_1"
                fi
            fi
        done
    fi
done
```

b) **bilatu_hasierak** scripta programatu honakoa egin dezan: argumentu bezala fitxategi hasierak jasoko ditu, eta bai uneko katalogoan baita azpikatalogoetan (maila bat jaitsez), hasiera bakoitzarekin hasten diren fitxategi guztien izenak idatzi beharko du pantailan.

Adibidez: **bilatu_hasierak lab arik fitx adib**

lab, arik, fitx edo **adib** hasiera duten fitxategi guztien izenak pantailan idatziko ditu.

2. ariketa [2 puntu]

a) UNIXeko Sarrera/Irteerako sistema-deiak erabiliz, hurrengo **funtzioa** kodetu behar da C lengoaian:

```
int fitxategi_zatia_pantailaratu(char *fitx_izena);
```

Funtzio honek honakoa egingo du: parametro bezala pasatako fitxategiaren tamaina (bytetan) 100 baino handiagoa bada, irteera estandarrean fitxategiaren edukiaren 80..100 byte tartean dauden karaktereak idatziko ditu, eta funtzioak 0 bueltatuko du. Aldiz, fitxategiaren tamaina 100 baino txikiagoa bada, edota erroreren bat gertatzen bada atzipenean, ez du ezer idatziko irteera estandarrean eta -1 bueltatuko du.

b) Aurreko ataleko funtzioa eta UNIXeko katalogoak atzitzeko liburutegi errutinak erabiliz, hurrengo **programa** kodetu behar da C lengoaian:

```
dir_patroia_zatia patroia
```

Programa honek irteera estandarrean, uneko katalogoan argumentu bakar bezala pasatako patroiaz hasten diren fitxategien zerrenda idatziko du, baldin eta bere tamaina (bytetan) 100 baino handiagoa bada. Fitxategi bakoitza lerro batean idatziko da, bere izena eta bere edukiaren 80..100 byte tartean dauden karaktereak idatziko direlarik.

Hona hemen irteera estandarrean idatzi beharrekoaren formatua:

```
fitx_izena_1:zattia_fitx_1  
fitx_izena_2:zattia_fitx_2  
...
```

Honako funtzio lagungarria kodetutzat ematen da, zuzenean erabiltzeko:

```
int hasten_da(char *fitx_izena, char *patroia);
```

Funtzio honek, lehen parametro bezala pasatako fitxategiaren izena bigarren parametro bezala pasatako patroiaz hasten bada 0 bueltatuko du, eta bestela -1.

3. ariketa [1,5 puntu]

UNIXeko *last* komandoak, sistemara egindako konexio guztien informazioa irteera estandarrean idazten du. Informazioa konexioen unearen arabera ematen du, berrienagatik zaharreneira sailkatuta. Konexio bakoitzeko lerro bat idazten du irteera estandarrean, besteak beste erabiltzailearen izena, konexioaren unea eta iraupena idazten direlarik. Horrela, erabiltzaile bat behin baino gehiago konektatu bada sistemara, egindako konexio beste lerro idatziko dira.

Aurrekoa jakinik, hurrengo **programa** kodetu behar da:

```
azken_n_konexioak n fitxategia
```

Programa honek bi argumentu jasotzen ditu, zenbaki bat (n) eta fitxategi baten izena, eta sistemara egindako azken n konexioei buruzko informazioa pasatako fitxategian idatziko du. Fitxategia existitzen ez bada, programak sortu egingo du.

Programa honen bi bertsio idaztea eskatzen da:

- a) Lehen bertsio bat UNIXeko *komando lerro bakar bat* erabiliz (suposatu $n = 100$ eta fitxategia *azken_100.txt* direla).
- b) Bigarren bertsio *multiprogramatu* bat C lengoaiatz, UNIXeko *komandoak* erabiliz (ikus bukaerako zerrenda) eta beharrezko iruditzen zaizkizun izenik gabeko buzoiez (*pipe*-ak) baliatuz.

(OHARRA: Ezin da “*exekutatu_programa*” funtzioa zuzenean erabili.)

4. ariketa [2,5 puntu]

Fakultateko kalkulu-zentroan, UNIX makina berri bat erosi dute kalkulu matematikoak burutzeko. Makina hau erabiltzeko, Sistema Eragileko ikasleoi Bezero/Zerbitzari sistema bat eraikitzea eskatu digute, Zerbitzariaren lana, eskaerak jaso eta bertan agertzen den kalkulu eskaera martxan ipintzea delarik. Zerbitzariaren buzoia **MBX_ZERB** izango da, eta eskaeraren formatua honakoa da:

```
struct eskaera {
    char bez_kodea[20];
    char pasahitza[20];
    char kalkulatzeko[256];
    char bez_buzoia[20];
}
```

Zerbitzariak, eskaera bat jasotzen duenean, bezeroak UNIX makina erabiltzeko baimena duen edo ez egiaztatu beharko du, eta horretarako hurrengo funtzioa erabiliko da (funtzio honen exekuzioa oso azkarra da):

```
int baimena(char *bez_kodea, char *pasahitza);
```

Funtzio honek 1 bueltatzen du erabiltzailea baimendua dagoenean, eta 0 baimendua ez dagoenean. Bezeroak baimenik ez badu, bere eskaera ez da zerbitzatuko eta erantzungo zaio ***struct erantzuna*** egiturarekin esanez ez duela baimenik; aldiz, baimena badu, bezeroari baimena duela erantzungo zaio ***struct erantzuna*** egiturarekin ere, eta erantzuna bidaliko zaio eskaera guztiz burutu denean.

```
struct erantzuna {
    char bez_kodea[20];
    int baimendua; //baimena badu=1, ez badu baimena=0
}
```

Kalkulu matematikoa burutzeko hurrengo ***programa*** dugu:

```
kalkulua_burutu bez_kodea kalkulatzeko
```

Kontuan hartu behar da UNIX makina honek gehienez **500 kalkulu** burutu ditzakeela modu konkurrentean.

Hurrengoa eskatzen da:

- Marras 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 (bezeroaren atala kodetuzat jotzen da, ez egin).
- Eskaera guztiz burutu denean bezeroari erantzuna bidali beharrean, orain eskaera zerbitzatzen hasten denean bidali nahi da. Adierazi ezazu zer aldaketa egin behar den (a) ataleko eskeman, eta marraztu eskema berria (ez da ezer kodetu behar, soilik aldaketak azaldu).

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 posiz, int nondik);
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 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
    }

```

st_dev:	unitatearen zenbakia
st_ino:	inodo zenbakia
st_mode:	modua
st_nlink:	lotura kopurua
st_uid:	jabearen identifikatzailea
st_gid:	taldearen identifikatzailea
st_size:	tamaina bytetan
st_atime:	azken atzipenaren data
st_mtime:	azken aldaketaren data
st_ctime:	sortu zeneko data

stat egitura

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);

```

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

```