5. Praktika

Multiprogramazioa: komunikazioa

Kontzeptuak

Prozesuen arteko komunikazioa eta sinkronizazioa. Mezu-trukea.

Urratsak

1 Sistema-dei berriak.

2 Sistema-deien ariketen programazioa.

3 Ariketen proba.

Dokumentazioa

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

UNIXeko sistema-deiak (komunikaziorako)

Buzoiak sortzea:

```
#include <sys/types.h>
#include <sys/stat.h>

int mknod(char *path, int modua, int dispositiboa);
int mkfifo(char *path, int modua);   /* liburutegiko errutina */
```

mknod sistema-deiaren bidez edozein motako fitxategiak sor daitezke, baina katalogoak, FIFOak eta fitxategi bereziak sortzeko bide bakarra da, **creat**-ek nahiz **open**-ek fitxategi arruntak soilik sortzen baitituzte. FIFOak edo buzoiak sortzeko erabiliz gero objektu horien gaineko beste zenbait sistema-deiren semantika aldatzen du:

write:	Datuak buzoian idazten ditu. Buzoia beteta baldin badago, prozesua blokeatu egingo da beste prozesu batek buzoi horretan **read** eragiketa burutu arte. Aurretik **open** edo **fcntl**-en bidez O_NDELAY ezarri bada beteta dagoen buzoi batean **write** egitean 0 balioa lortuko da eta prozesua ez da blokeatuko.
read:	Datuak irakurtzen ditu buzoitik heldu diren ordena berean eta, irakurtzen dituen heinean kentzen ditu. Buzoia hutsik baldin badago prozesua blokeatu egingo da gutxienez karaktere bat egon arte, bi salbuespenetan izan ezik: idazketaren deskribatzailea itxita dagoenean edo O_NDELAY ezarri denean irakurketaren deskribatzailean. Bi kasu hauetan prozesua ez da blokeatuko eta 0 balioa jasoko da.
open:	Irakurketarako edo idazketarako soilik irekitzen denean, prozesua blokeatuko da beste batek eragiketa osagarrirako irekitzen ez duen bitartean, O_NDELAY ezartzen ez denean. Beraz zerbitzarietan O_RDWR erabiltzea gomendatzen da.
close:	Idazketaren deskribatzailean egiten bada, deskribatzailea libratzeaz gain, buzoian EOF idazten du, azken idazlea baldin bada. Irakurketaren deskribatzaile guztiak ixten badira aldiz, idazketaren deskribatzailean **write** egitean errorea itzuliko du.

```
main()   /* who | wc */
{  char str[256];
   sprintf(str, "buzoiaren_izena");
   if (mknod(str, S_IFIFO|0666, 0) < 0) errore("mknod");
   switch (fork()) {
      case -1: errore("fork");
      case  0: /* 1. umea: who */
               if (close(1) == -1) errore("close");
               if (open(str, O_WRONLY) != 1) errore("open");
               execlp("who", "who", NULL);
               errore("execlp");
   }
   switch (fork()) {
      case -1: errore("fork");
      case  0: /* 2. umea: wc */
               if (close(0) == -1) errore("close");
               if( open(str, O_RDONLY) != 0) errore("open");
               execlp("wc", "wc", NULL);
               errore("execlp");
   }
   while (wait(NULL) !=- 1) ;
   unlink(str);
}
```

Izendun buzoien adibidea (*mknod*)

***Pipe*-ak (izenik gabeko buzoiak) sortu eta irekitzea**:

```
int pipe(int pfd[2]);
```

Dei honek komunikaziorako kanal bat sortzen du. Fitxategi-deskribatzaileen balioak erreferentziaz pasatzen dira emaitza bikoitza jaso ahal izateko. **pfd[0]**n irakurketarako fitxategiaren deskribatzailea kokatuko da eta **pfd[1]**ean idazketarako deskribatzailea. **open** deia bi aldiz egiteak eta **pipe** egiteak antzeko eragina dute. *Pipe*-en gaineko eragiketen semantika buzoienarena da.

Bestelakoak:

```
int dup(int fd);
```

dup deiaren ondorioz parametro bezala pasatako kanala bikoizten da, kanal berri bat sortuz. Kanal berria aurrekoaren kopia zehatza da. Unix-ek zera ziurtatzen du: libre daudenen artean indizerik txikieneko kanala esleitzen duela; hortaz, **dup** egin aurretik **close(0)** egiten bada sarrera lehenetsian bikoiztuko da kanala. Gauza bera egin daiteke beste edozein kanalekin. Fitxategiekin edo izendun buzoiekin hauxe bera egin daiteke **open** deiaren bitartez baina *pipe*-kin ez, azken hauek izenik ez dutelako.

```
main()   /* who | wc */
{
   int pfd[2];

   if (pipe(pfd) == -1) errore("pipe");

   switch (fork()) {
      case -1: errore("fork");
      case  0: /* 1. umea: who */
               if (close(1) == -1) errore("close");
               if (dup(pfd[1]) != 1) errore("dup");
               if ((close(pfd[0]) == -1) ||
                   (close(pfd[1]) == -1)) errore("close");
               execlp("who", "who", NULL);
               errore("execlp");
   }

   switch (fork()) {
      case -1: errore("fork");
      case  0: /* 2. umea: wc */
               if (close(0) == -1) errore("close");
               if (dup(pfd[0]) != 0) errore("dup");
               if ((close(pfd[0]) == -1) ||
                   (close(pfd[1]) == -1)) errore("close");
               execlp("wc", "wc", NULL);
               errore("execlp");
   }

   if ((close(pfd[0]) == -1) || (close(pfd[1]) == -1)) errore("close");

   while(wait(NULL)!=-1) ;
}
```

Izenik gabeko buzoien adibidea (*pipe, dup*)

ARIKETAK – Multiprogramazioa (Komunikazioa)

1.- Komando-interpretatzaile berri bat egin dugu aurreko praktikan eta **`history`** izeneko utilitate berri bat erantsi nahi diogu aipatutako interpretatzaileari. Honen helburua hauxe da: interpretatzailera iristen diren komando-lerroak gordetzea, komando-lerro berri bat "!" karaktereaz hasten denean, karaktere horren ondoren dauden karaktereez hasitako azken komando-lerroa berregikari dadin. Adib. honako lerro hauen sekuentzia emanda:

```
[1] jaurti_history> who acaf0151
[2] jaurti_history> rm tmp1 tmp2
[3] jaurti_history> !w
```

[3] komando-lerroaren eragina zera da: [1] komando-lerro osoaren exekuzioaren errepikapena. Azken 24 lerroak metatzen dira eta lerroa "!" ez den karaktereaz hasten denean komando-lerroa exekutatuko da norma-normal.

`jaurti_history` utilitate berri hau programatu behar duzu honako hau kontuan hartuz:

- **`history`** komandoa sakatzen denean inprimatuko dira azken 24 komando-lerroak.
- utilitate berria hasieran jarriko da martxan, komando-lerroak iragaz ditzan **`jaurti`** programari bidaliz.
- beraz **`jaurti`** prozesu umea izango da eta bere sarrera estandarra berbideratuko da.
- **`jaurti_history`** bukatuko da sarreratik EOF jasotzen denean edo `exit` komandoa irakurtzen denean.

Laguntza:

azken 24 komandoak kudeatzeko honako funtzio hauek ditugu definituta (history_taula.h) fitxategian:

```
extern void gorde_kom(char *kom);
extern char *bilatu_kom(char *has);
extern void inprimatu_kom();
```

2.- Idatz ezazu **`kontatu_filtro_lerroak`** programa bat zeinak irteera estandarrean idatziko duen hitz konkretu bat duten lerro kopurua. Testu fitxategia eta bilatu beharreko hitza izango dira programa honen argumentu bakarrak. Programari dei egiteko modua ondokoa izango da:

```
kontatu_filtro_lerroak  fitxategia  hitza
```

Programa honen bi bertsio desberdin egitea proposatzen da:

a) Lehen bertsio bat UNIX-eko komando-lerro bakar bat erabiliz, adibide bezala **`datuak.txt`** eta **`FAKULTATEA`** parametroak erabiliz.
b) Bigarren bertsio multiprogramatu bat, UNIX-eko utilitate estandarrak erabiliz, eta beharrezko ikusten dituzun komunikazio mekanismoez baliatuz. Soluzio hau inplementatzeko ezingo da `exekutatu_programa` errutina erabili, eta **`fork`** eta **`exec`** erabili beharko dira zuzenean.
c) Hirugarrengo bertsio multiprogramatua egin, oraingo honetan **`exekutatu_programa`** errutina erabiliz. Aztertu aurreko bertsioarekin dauden desberdintasunak eta ulertu.

3.- Idatz ezazu **`erabiltzaile_konektatuak_sailkatuta`** programa bat zeinak irteera estandarrean idatziko duen sistemara konektatuta dauden erabiltzaileen zerrenda alfabetikoki sailkatuta.

Programa honen bi bertsio desberdin egitea proposatzen da:

a) Lehen bertsio bat UNIXeko ***komando lerro bakar bat*** erabiliz.
b) Bigarren bertsio ***multiprogramatu*** bat, C lengoaiaz, UNIXeko ***utilitate estandarrak*** erabiliz, eta beharrezko edo egokien iruditzen zaizkizun komunikazio mekanismoez baliatuz.

4.- Idatz ezazu **`hasierako_bostak`** programa bat, zeinak irteera estandarrean idatziko dituen sisteman konektatuta dauden erabiltzaileen artean lehenengo bostak orden alfabetikoan.

Programa honen bi bertsio idaztea eskatzen da:

a) Lehen bertsio bat UNIXeko ***komando lerro bakar bat*** erabiliz.
b) Bigarren bertsio ***multiprogramatu*** bat, C lengoaiaz, UNIXeko ***komandoak*** erabiliz eta beharrezko iruditzen zaizkizun komunikazio mekanismoez baliatuz.

5.- Idatz ezazu **`acaf_konektatuak`** programa bat *`ikasleak.txt`* fitxategian idatziko duena sisteman konektatuta dauden gure klaseko erabiltzaileak (hau da, bere erabiltzaile izenean “acaf” testu-katea dutenak).

Programa honen bi bertsio idaztea eskatzen da:

a) Lehen bertsio bat UNIXeko ***komando lerro bakar bat*** erabiliz.
b) Bigarren bertsio ***multiprogramatu*** bat, C lengoaiaz, UNIXeko ***komandoak*** erabiliz eta beharrezko iruditzen zaizkizun komunikazio mekanismoez baliatuz. Soluzio hau inplementatzeko ezingo da `exekutatu_programa` errutina erabili, eta **`fork`** eta **`exec`** erabili beharko dira zuzenean.
c) Hirugarrengo bertsio multiprogramatua egin, oraingo honetan **`exekutatu_programa`** errutina erabiliz. Aztertu aurreko bertsioarekin dauden desberdintasunak eta ulertu.

6.- Programa bat egitea eskatu digute, *`foro_laguntzaile`* izenekoa. Programa honek fitxategi baten izena jasoko du, eta irteera estandarrean foro konkretu bati (programaren bigarren argumentua) dagokion mezuak idatziko ditu, eta beste mezu guztiak hirugarren argumentu gisa pasatako buzoian. Horretarako *grep* programaz baliatuko gara (gure programaren ume prozesu gisa). Programa honek patroi konkretu bati jarraitzen dioten lerroak ematen dizkigu (alegia, foroaren izena gure kasuan). Gainera, “-v” adierazlearekin, *grep* programak patroia jarraitzen ez duten lerroak ematen dizkigu. Mezu bakoitza lerro bakarra okupatzen du fitxategian, honako egitura duelarik: *foroa: mezuaren testua*.

Idatz ezazu programa hau: *`foro_laguntzaile fitxategia foroa buzoia`*