



НЕГІЗГІ ЕНГІЗУ- ШЫҒАРУ ОПЕРАЦИЯЛАРЫ

Мазмұны

- Файлдарды құру және ашу
- Файлдарды жабу
- Файлдарды оқу және жазу
- Файлдардағы ағымдағы енгізу-шығару орнын жылжыту

Файлды құру:

`creat()`

Linux-та төмен деңгейлі енгізу-шығару механизмдерін егжей-тегжейлі қарастырмас бұрын, файл режимі ұғымын анықтаймыз. Әр файлға Linux-та 16 биттік тізбек байланған, оны үш топқа бөлуге болады:

1. 0-8 биттері: негізгі қол жетімділік құқықтары.
2. 9-11 биттері: кеңейтілген қол жетімділік.
3. 12-15 биттері: файл түрі.

Бұл тізбек файл режимі деп аталады.



Файл түрі ондағы ақпараттың сипатын (мәтін, сурет, музыка және т.б.) емес, төменгі деңгейдегі енгізу-шығару ерекшеліктерін анықтайды. Осы өлшем бойынша ядро барлық файлдарды келесі түрлерге бөледі:

- ҚАРАПАЙЫМ ФАЙЛ

бұл файлдар деректерді сақтауға арналған. Бинарлар мен кітапханалар да қарапайым файлдарға жатады.

- СИМВОЛДЫҚ ҚҰРЫЛҒЫ

- БЛОК ҚҰРЫЛҒЫСЫ

- КАТАЛОГ

- СИМВОЛДЫҚ СІЛТЕМЕ

басқа файлдарды көрсетеді. Ол -s жалаушасы бар ln бағдарламасы арқылы жасалады.

- FIFO АРНАСЫ

FIFO арналары (First In First Out) процессаралық өзара әрекеттесуге қатысады.

- СОКЕТ

гетерогенді жүйелер арасындағы өзара әрекеттесудің әмбебап құралы.



LINUX-КӨП ҚОЛДАНУШЫ ОПЕРАЦИЯЛЫҚ ЖҮЙЕ.
ПАЙДАЛАНУШЫЛАРДЫҢ ФАЙЛДАРҒА ҚАТЫСТЫ
ӨКІЛЕТТІКТЕРІН БӨЛУ ҮШІН ЯДРОДА ӘР ФАЙЛДЫҢ
ИЕСІ (USER, OWNER) ЖӘНЕ ТОБЫ (ТОБЫ) БАР ҚОЛ
ЖЕТКІЗУ ҚҰҚЫҒЫ ТҰЖЫРЫМДАМАСЫ ЖҮЗЕГЕ
АСЫРЫЛАДЫ. ФАЙЛДЫҢ ИЕСІ ӘРҚАШАН БІР, АЛ
ТОПТА БІРНЕШЕ ПАЙДАЛАНУШЫ БОЛУЫ МҮМКІН.
СОНЫМЕН ҚАТАР, БАСҚА ПАЙДАЛАНУШЫ (OTHERS)
ТҮСІНІГІ ДЕ ЕРЕКШЕЛЕНЕДІ, ЯҒНИ.СУПЕР
ПАЙДАЛАНУШЫ (ТҮБІР) ҚҰҚЫҒЫНА ИЕ ЕМЕС, ИЕСІ
ЕМЕС ЖӘНЕ КӨРСЕТІЛГЕН ТОПҚА КІРМЕЙДІ.

САНАМАЛАНҒАН САНАТТАРДЫҢ ӘРҚАЙСЫСЫ ҮШІН (ИЕСІ, ТОБЫ, БАСҚАЛАРЫ) ТӨМЕНДЕ САНАМАЛАНҒАН ҮШ ӨЛШЕМШАРТТЫ НЕГІЗГЕ АЛА ОТЫРЫП, ЖЕКЕ ҚОЛ ЖЕТКІЗУ ҚҰҚЫҒЫ БЕЛГІЛЕНЕДІ.

READ

Оқу құқығы— пайдаланушыға файлдан деректерді оқуға мүмкіндік береді. Егер файл каталог болса, онда оқу құқығы оның мазмұнын көру мүмкіндігі ретінде түсіндіріледі.

WRITE

Жазу құқығы— пайдаланушыға файлға деректерді жазуға мүмкіндік береді. Егер файл каталог болса, онда жазу құқығы осы каталогтағы файлдарды құру, жою және атын өзгерту мүмкіндігі ретінде түсіндіріледі.

EXECUTE

Орындау құқығы — пайдаланушыға файлды іске қосуға мүмкіндік береді. Егер файл каталог болса, онда орындау құқығы пайдаланушыға осы каталогқа "кіруге", яғни оны ағымдағы (ағымдағы) етіп жасауға мүмкіндік береді.



Бит	Права	Константа
0	Выполнение для остальных пользователей	S_IXOTH
1	Запись для остальных пользователей	S_IWOTH
2	Чтение для остальных пользователей	S_IROTH
3	Выполнение для группы	S_IXGRP
4	Запись для группы	S_IWGRP
5	Чтение для группы	S_IRGRP
6	Выполнение для владельца	S_IXUSR
7	Запись для владельца	S_IWUSR
8	Чтение для владельца	S_IRUSR

КІРУ ҚҰҚЫҒЫНЫҢ БИТТЕРІН
ЖӘНЕ ОЛАРҒА СӘЙКЕС
ТҰРАҚТЫЛАРДЫ ТАҒАЙЫНДАУ

0-ден 8-ге дейінгі файл режимінің биттері иесіне, топқа және басқа пайдаланушыларға қол жеткізу құқығын сипаттайды. Кіру құқығын көрсету үшін бағдарламашылар арнайы аталған режим тұрақтыларын қолдана алады, оларды биттік ажырату (немесе) операциясымен байланыстырады. Режим тұрақтыларына қол жеткізу үшін Бағдарламаға `sys/stat.h` тақырып файлын қосу керек.

МЫСАЛ

Бағдарламада "иесіне оқу және жазу", сондай-ақ "топқа оқу" құқықтарын орнату керек делік. Бұл осылай жасалады:

`S_IRUSR | S_IWUSR | S_IRGRP`

Енді файл дескрипторын қайтару арқылы көрсетілген режиммен файлды құратын және ашатын `create ()` жүйелік қоңырауын қарастырайық. `fcntl.h` тақырып файлында `create()` функциясының прототипі жарияланды келесідей:

`int creat (const char * FILENAME, mode_t MODE);`

Бірінші аргумент (`FILENAME`) — файл атауы. Екінші аргумент (`MODE`) — файл режимі.

Айта кету керек, дескрипторлар кестесінде теріс мәндер болмауы мүмкін. Егер `creat()` жүйелік қоңырауы `-1` қайтарылса, бұл файлды құру және ашу мүмкін еместігін білдіреді.

```
#INCLUDE <STDIO.H>
#include <FCNTL.H>
#include <SYS/TYPES.H>
#include <SYS/STAT.H>
#include <UNISTD.H>
INT MAIN (INT ARGV, CHAR ** ARGV)
{
    INT FD;
    MODE_T FMODE = S_IRUSR;
    IF (ARGV < 2) {
        FPRINTF (STDERR, "TOO FEW
        ARGUMENTS\n");
        RETURN 1;
    }
    FD = CREAT (ARGV[1], FMODE);
    IF (FD == -1) {
        FPRINTF (STDERR,
        "CANNOT CREATE FILE (%S)\n",
        ARGV[1]);
        RETURN 1;
    }
    CLOSE (FD);
    RETURN 0;
}
```

Енді иесі үшін көрсетілген аты мен оқу құқығы бар файл жасайтын бағдарламаны жазамыз

Бағдарламада алдымен файл дескрипторын орналастыру үшін `creat()` жүйелік қоңырауы сәтті аяқталған кезде бүтін типті айнымалы жасалады. Жалғыз `s_irsr` тұрақтысы `fmode` айнымалысына енгізіледі, ол файл режимінің 8-ші битінің бірлігіне орнатылады (иесі үшін оқылады). Содан кейін `creat ()` қоңырауы келеді, ол сәтсіздікке ұшыраған жағдайда -1 қайтарады.



Берілген бағдарлама өте қарапайым, бірақ ең қызықтысы әлі келмейді. Бірінші іске қосу кезінде бәрі жоспар бойынша жүреді:

```
$ GCC -O CREAT1 CREAT1.C
```

```
$ ./creat1 myfile
```

```
$ ls -l myfile
```

```
-r----- 1 nnivanov nnivanov 0 2011-05-05 11:50 myfile
```

Шынында да, қажетті қол жетімділік құқықтары бар нөлдік ұзындықтағы файл жасалды.

Бірақ егер сіз creat1 бағдарламасын тағы бір дәлелмен іске қоссаңыз, сіз тосынсый аласыз:

```
$ ./CREAT1 MYFILE
```

```
Cannot create file (myfile)
```

Бұл файл қазірдің өзінде бар емес, бірақ кіру құқығы тек оқуға арналған. Егер файл бар болса, creat() оны ағымдағы кіру құқығын өзгертпестен тазартады. Бірақ бағдарламаны бірінші рет шақырған кезде файл "тек оқу үшін" қатынасу құқығымен жасалғандықтан, creat() қоңырауы оны өшіре алмады (тазалауға ештеңе болмаса да) және сәтсіз аяқталды.

Бұл жағдайды түзету оңай:

```
$ CHMOD U+W MYFILE
```

```
$ ls -l myfile
```

```
-rw----- 1 nnivanov nnivanov 0 2011-05-05 11:50 myfile
```

```
$ ./creat1 myfile
```

```
$ ls -l myfile
```

```
-rw----- 1 nnivanov nnivanov 0 2011-05-05 11:52 myfile
```

Файлға кіру құқығы өзгермегеніне назар аударыңыз. Сонымен қатар creat() бар файлдарды тазалайды.

ФАЙЛДЫ АШУ: OPEN()

Файлды белгілі бір жолмен open () жүйелік қоңырау арқылы ашуға болады.

Open() — нің жағымды ерекшелігі-файлды ашу процесін бақылайтын жалаушаларды орнату мүмкіндігі.

Файлды ашу жалаулары (режим жалаулары сияқты) - бұл биттік ажырату (немесе) операциясы арқылы қосыла алатын символдық тұрақтылардың жиынтығы. Осы жалаушаларды және open () жүйелік қоңырауды пайдалану үшін бағдарламаға fcntl.h тақырып файлын қосу қажет емес.

ФАЙЛДЫ АШУ ЖАЛАУШАЛАРЫН БІРІКТІРУДІҢ НӘТИЖЕСІ
ҚАРАПАЙЫМ БҮТІН САН БОЛЫП ТАБЫЛАДЫ.

- O_RDONLY-файлды тек оқу үшін ашу;
- O_WRONLY-тек жазу үшін файлды ашу;
- O_RDWR-оқу және жазу үшін файлды ашу;
- O_CREAT-егер жоқ болса, файл жасау;
- O_TRUNC-егер бар болса, файлды тазалау;
- O_APPEND-бар файлға қосу;
- O_EXCL-егер бар болса, файлды ашпау (o_creat жалаушасымен қолданылады).



fcntl.h тақырып файлында жарияланған
open () жүйелік қоңырауы, екі прототипі бар:

```
int open (const char * FILENAME, int FLAGS, mode_t MODE);  
int open (const char * FILENAME, int FLAGS);
```



FILENAME және MODE аргументтері create () жүйелік қоңырау дәлеліне толығымен ұқсас. Open () екінші прототипі әдетте режимді көрсету артық болған кезде "тек оқу" режимінде файлдарды ашу үшін қолданылады. FLAGS аргументі-бұл ұрылған немесе операциямен біріктірілген ашылу жалауларының жиынтығы. Айта кету керек, кейбір жалаулар бір-біріне қосыла алмайды. Мұнда логикалық ойлау керек. Мысалы, O_RDONLY және O_WRONLY жалауларын біріктіруге болмайды, өйткені олар бір-біріне қайшы келеді.



КӨБІНЕСЕ CREAT() ЖӘНЕ OPEN () ЖҮЙЕЛІК ҚОҢЫРАУЛАРЫНЫҢ MODE АРГУМЕНТІ РЕТІНДЕ ОПЕРАЦИЯМЕН БІРІКТІРІЛМЕГЕН (НЕМЕСЕ) АТАЛҒАН ТҰРАҚТЫЛАРДЫ ЕМЕС, СЕГІЗДІК САНДЫҚ ЖҮЙЕДЕГІ САНДАРДЫ КӨРСЕТЕДІ. БҰЛ КӨРІНІС ҚОЛ ЖЕТКІЗУ ҚҰҚЫҒЫНЫҢ СЕГІЗДІК КӨРІНІСІ ДЕП АТАЛАДЫ. МҮМКІН СІЗ БҰЛ ТӘСІЛДІ ЕШҚАШАН ҚОЛДАНБАЙТЫН ШЫҒАРСЫЗ, БІРАҚ ҚОЛ ЖЕТКІЗУ ҚҰҚЫҒЫН СЕГІЗ ФОРМАДА КӨРУ ҚАБІЛЕТІ БАСҚА АДАМДАРДЫҢ БАҒДАРЛАМАЛАРЫН ОҚУ КЕЗІНДЕ ПАЙДАЛЫ БОЛУЫ МҮМКІН. СОНЫМЕН ҚАТАР, ҚОЛ ЖЕТКІЗУ ҚҰҚЫҒЫН СЕГІЗДІК ТҮСІНДІРУ ЖҮЙЕЛІК ӘКІМШІЛЕРДІҢ ШЕҢБЕРІНДЕ КЕҢ ТАРАЛҒАН.

ФАЙЛ РЕЖИМІНІҢ 9 БИТІН АЛАТЫН КІРУ ҚҰҚЫҒЫН ЖАЗУ ҮШІН ҮШ СЕГІЗДІК САН ЖЕТКІЛІКТІ. БИТТЕР ОҢНАН СОЛҒА ҚАРАЙ ЕСЕПТЕЛЕТІНІН ЕСТЕ ҰСТАҒАН ЖӨН (LITTLE ENDIAN). ҚОЛ ЖЕТІМДІЛІК БИТТЕРІНІҢ СЕГІЗДІК ЖАЗБАСЫ ЫҢҒАЙЛЫ, ӨЙТКЕНІ ӘР СЕГІЗДІК САН ҮШ БИТТІҢ ТІЗБЕГІНЕ СӘЙКЕС КЕЛЕДІ.

МҰНЫҢ ІС ЖҮЗІНДЕ ҚАЛАЙ ЖҰМЫС ІСТЕЙТІНІН
ТЕКСЕРУ ҮШІН АЛДЫМЕН ОНДЫҚ КӨРІНІСТЕ
КІРУ ҚҰҚЫҒЫН КӨРСЕТЕТІН ШАҒЫН
БАҒДАРЛАМА ЖАЗАМЫЗ

НЕ БОЛҒАНЫН ТЕКСЕРЕМІЗ:

```
$ GCC -O MYMASK MYMASK.C
$ ./MYMASK
420
```

СОНЫМЕН, "МЕНШІК ИЕСІ ҮШІН ОҚУ ЖӘНЕ
ЖАЗУ" ЖӘНЕ "ТОП ПЕН БАСҚА
ПАЙДАЛАНУШЫЛАР ҮШІН ОҚУ" ҚОЛ
ЖЕТІМДІЛІК ҚҰҚЫҚТАРЫ 420 САНЫНА СӘЙКЕС
КЕЛЕДІ. ЕНДІ БІЗ БҰЛ САНДЫ СЕГІЗДІК
САНДЫҚ ЖҮЙЕГЕ АУДАРАМЫЗ:

```
$ BC -Q
obase=8
420
644
quit
```

644 ШЫҚТЫ.
АЛЫНҒАН
САНДЫ
ЕКІЛІК
САНДАР
ЖҮЙЕСІНЕ
АУДАРАМЫЗ:

```
$ BC -Q
ibase=8
obase=2
644
110100100
quit
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
int main (void)
{
    mode_t test_mode = S_IRUSR | S_IWUSR | S_IRGRP
    | S_IROTH;
    printf ("%d\n", test_mode);
    return 0;
}
```

ЕНДІ АЛЫНҒАН ТІЗБЕКТІ ОҢНАН СОЛҒА ҚАРАЙ 0-ДЕН 8-ГЕ ДЕЙІН
САНАҢЫЗ. БҰЛ ҚОЛ ЖЕТКІЗУ ҚҰҚЫҒЫНЫҢ БИТТЕРІ. СОНЫМЕН
ҚАТАР, ЕГЕР СІЗ ОСЫ БИТТЕРДІ LS БАҒДАРЛАМАСЫ ШЫҒАРАТЫН
ӘДЕТТЕГІ RWXRWXRWX ШАБЛОНЫНА САЛСАҢЫЗ, БАРЛЫҚ
НӨЛДЕРДІ СЫЗЫҚШАЛАРМЕН АЛМАСТЫРСАҢЫЗ, ҚОЛ ЖЕТКІЗУ
ҚҰҚЫҒЫН ОҚУҒА ЫҢҒАЙЛЫ ТҮРДЕ АЛАМЫЗ:
RW-R--R--

СОНЫМЕН, СЕГІЗДІК ҚОЛ ЖЕТІМДІЛІК ҮШ САННАН ТҰРАДЫ:

- бірінші сан-иесіне қол жеткізу құқығы;
- екінші сан-топқа кіру құқығы;
- үшінші сан - қалған пайдаланушыларға қол жеткізу құқығы.

ЕГЕР СІЗ ӘР САННЫҢ НЕНІ БІЛДІРЕТІНІН ЕСТЕ САҚТАСАҢЫЗ, ОНДА СЕГІЗДІК ҚҰҚЫҚТАРМЕН ЖҰМЫС ЕШҚАНДАЙ ҚИЫНДЫҚ ТУҒЫЗБАЙДЫ (ЖАҚШАДА ЕКІЛІК МӘНДЕР БЕРІЛГЕН):

- 0-қол жеткізу құқығының болмауы (000);
- 1-Тек орындау (001);
- 2-тек жазба (010);
- 3-жазу және орындау (011);
- 4-тек оқу (100);
- 5-оқу және орындау (101);
- 6-оқу және жазу (110);
- 7-оқу, жазу және орындау (111).



СЕГІЗДІК ҚОЛ ЖЕТІМДІЛІК ҚҰҚЫҚТАРЫН CHMOD БАҒДАРЛАМАСЫМЕН ДЕ ПАЙДАЛАНУҒА БОЛАДЫ:

```
$ TOUCH MYFILE
```

```
$ ls -l myfile
```

```
-rw-r--r-- 1 nnivanov nnivanov 0 2011-05-05 12:02 myfile
```

```
$ chmod 600 myfile
```

```
$ ls -l myfile
```

```
-rw----- 1 nnivanov nnivanov 0 2011-05-05 12:02 myfile
```



```
#INCLUDE <STDIO.H>
#include <fcntl.h>
#include <unistd.h>
int main (int argc, char ** argv){
    int fd;
    if (argc < 2) {
        fprintf (stderr, "Too few
arguments\n");
        return 1;
    }
    fd = open (argv[1], O_WRONLY | O_CREAT |
O_TRUNC, 0640);
    if (fd == -1) {
        fprintf (stderr, "Cannot create file
(%s)\n",
argv[1]);
        return 1;
    }
    close (fd);
    return 0;
}
```

ЕҢ АЛДЫМЕН, CREAT() ЖҮЙЕЛІК
ҚОҢЫРАУЫ O_WRONLY, O_CREAT
ЖӘНЕ O_TRUNCТІҢ АШЫЛУ
ЖАЛАУЛАРЫМЕН ШАҚЫРЫЛҒАН
OPEN () - МЕН БІРДЕЙ ЕКЕНІН АТАП
ӨТКЕН ЖӨН. ЕГЕР СЕНБЕСЕҢІЗ,
LINUX ЯДРОСЫНЫҢ БАСТАПҚЫ
КӨЗДЕРІНЕ FS/OPEN.C ФАЙЛЫН
ҚАРАҢЫЗ.

Бұл бағдарламаға sys/stat.h және sys
/ types.h файлдары қосылмайды,
өйткені біз режим тұрақтыларын да,
mode_t түрін де қолданбаймыз.

● Файлды жабу:

close()

Close () жүйелік қоңырауы файл дескрипторын босатады. Егер деректер жазбасын Буферлеу қолданылса, close() ядроға синхрондау сұрауын жібереді. Бірақ бұл деректер дереу медиаға жазылады дегенді білдірмейді. Close () жүйелік қоңырауы unistd.h файлында келесідей жарияланды:

```
int close (int FD);
```

Алдыңғы мысалдарда біз close() бір нәрсені қайтара алатындығын байқамадық. Файлды жабу кезіндегі қателер өте сирек кездеседі, бірақ олар кейде болады. Сондықтан close () жүйелік қоңырауы сәтті аяқталған кезде 0 немесе қате болған жағдайда -1 қайтарады.




```
#INCLUDE <STDIO.H>
#include <fcntl.h>
#include <unistd.h>
int main (int argc, char ** argv)
{
    int fd;
    if (argc < 2) {
        fprintf (stderr, "Too few arguments\n");
        return 1;
    }
    fd = open (argv[1], O_WRONLY | O_CREAT | O_EXCL, 0644);
    if (fd == -1) {
        fprintf (stderr, "Cannot create file (%s)\n",
            argv[1]);
        return 2;
    }
    if (close (fd) == -1) {
        fprintf (stderr, "Cannot close file "
            "with descriptor=%d\n", fd);
        return 3;
    }
    if (close (fd) == -1) {
        fprintf (stderr, "Cannot close file "
            "with descriptor=%d\n", fd);
        return 4;
    }
    return 0;
}
```

БҰЛ БАҒДАРЛАМА АПАТТЫҚ БОЙЫНША АНЫҚТАУ. ҚАТЕЛЕРДІ ӨҢДЕУДІҢ БАРЛЫҚ БЛОКТАРЫ RETURN НҰСҚАУЛАРЫН ӘРТҮРЛІ САНДЫҚ МӘНДЕРГЕ БЕРЕТІНІНЕ НАЗАР АУДАРЫҢЫЗ. СІЗ MAIN() ФУНКЦИЯСЫ ҚАЙТАРУ КОДЫ ДЕП АТАЛАТЫН БАҒДАРЛАМАНЫҢ АЯҚТАЛУ КҮЙІН ҚАЙТАРАТЫНЫН БІЛЕСІЗ. ТАРИХИ ТҰРҒЫДАН АЛҒАНДА, НӨЛДІК ҚАЙТАРУ КОДЫ БАҒДАРЛАМАНЫҢ СӘТТІ АЯҚТАЛҒАНЫН, АЛ НӨЛДІК ЕМЕС КОД ҚАТЕ ТУРАЛЫ БІЛДІРЕДІ. КЕЛЕСІ ПӘРМЕН СІЗГЕ СОҢҒЫ АЯҚТАЛҒАН БАҒДАРЛАМАНЫҢ ҚАЙТАРУ КОДЫН БІЛУГЕ МҮМКІНДІК БЕРЕДІ:

```
$ ECHO $?
```

БІЗДІҢ МЫСАЛДА ЕКІ CLOSE () ТЕКСЕРУ БЛОГЫ БІРДЕЙ ХАБАРЛАМАНЫ КӨРСЕТЕДІ. БІРАҚ БІЗ ҚАЗІР ҚАТЕНІҢ ҚАЙ ЖЕРДЕ БОЛҒАНЫН БІЛЕ АЛАМЫЗ:

```
$ ./CLOSETWICE MYFILE
```

```
CANNOT CLOSE FILE WITH DESCRIPTOR=3
```

```
$ ECHO $?
```

CLOSETWICE БАҒДАРЛАМАСЫНДА ТАҒЫ БІР "ТҰЗАҚ"БАР. АЛДЫҢҒЫ ІСКЕ ҚОСУМЕН БІРДЕЙ ФАЙЛ АТАУЫН КӨРСЕТЕ ОТЫРЫП, ОСЫ БАҒДАРЛАМАНЫ ҚАЙТАДАН ІСКЕ ҚОСЫП КӨРІҢІЗ:

```
$ ./CLOSETWICE MYFILE
```

```
CANNOT CREATE FILE (MYFILE)
```

```
$ ECHO $?
```

```
2
```

БҰЛ ЖОЛЫ ІС ЖАБЫЛҒАНҒА ДЕЙІН ЖЕТКЕН ЖОҚ. БАРЛЫҚ КІНӘ O_EXCL ТУЫ БОЛДЫ, ОЛ БҰРЫННАН БАР ФАЙЛДЫ ҚАЙТА АШУҒА МҮМКІНДІК БЕРМЕЙДІ.

ФАЙЛДАРДЫ ОҚУ: READ()

Файлдарды оқу үшін unistd.h файлында жарияланған read () жүйелік қоңырауы қызмет етеді келесідей:

```
ssize_t read (int FD, void * BUFFER, size_t SIZE);
```

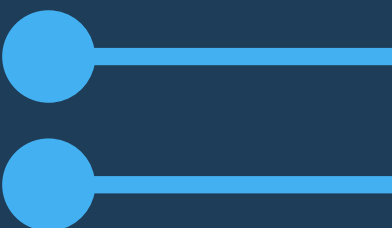
Бұл жүйелік қоңырау FD дескрипторы бар файлдан SIZE байтын оқуға тырысады. Оқылған ақпарат буферге енгізіледі. Қайтару мәні - нақты оқылған байттардың саны.

Size_t аргументі-бұл қол қойылмаған тұтас тип, оның өлшемі іске асыруға байланысты; ssize_t-ақ бүтін, бірақ қазірдің өзінде белгі түрі. Ssize_t нұсқауы қате болған жағдайда read() -1 қайтаратындығына байланысты. Ағымдағы енгізу-шығару орны файлдың соңына жеткенде, read () 0 қайтарады.

Буфер ретінде белгілі көлемдегі кез-келген жад блогы әрекет ете алады. Бірақ көбінесе read () жүйелік қоңырауында жолдар қолданылады.

```
INCLUDE <STDIO.H>
#include <fcntl.h>
#include <unistd.h>
int main (int argc, char ** argv)
{
    int fd;
    char ch;
    if (argc < 2) {
        fprintf (stderr, "Too few arguments\n");
        return 1;
    }
    fd = open (argv[1], O_RDONLY);
    if (fd == -1) {
        fprintf (stderr, "Cannot open file (%s)\n",
            argv[1]);
        return 1;
    }
    while (read (fd, &ch, 1) > 0) printf ("%c", ch);
    if (close (fd) == -1) {
        fprintf (stderr, "Cannot close file "
            "with descriptor=%d\n", fd);
        return 1;
    }
    return 0;
}
```

Сонымен, char типіндегі айнымалы мекен-жай буфер ретінде беріліп, бір Байт оқылды. Мұндай схема қарапайым, бірақ өте тиімсіз. Әр таңбаны оқу үшін read () жүйелік қоңырауына жүгіну бағдарламаны айтарлықтай баяулатады. Елестетіп көріңіз: 1 Кбайт файлды оқу үшін бағдарлама ядроға 1024 рет жүгінеді! Сондықтан файлдарды жеткілікті үлкен блоктармен оқыған дұрыс. Әдетте осы мақсаттар үшін бірнеше килобайт буфер жасалады.



```

#include <STDIO.H>
#include <FCNTL.H>
#include <UNISTD.H>
#include <SYS/TYPES.H>
#define BUF_SIZE 4096
CHAR BUFFER[BUF_SIZE+1];
INT MAIN (INT ARGV, CHAR ** ARGV)
{
    SSIZE_T BYTES;
    INT FD;
    IF (ARGV < 2) {
        FPRINTF (STDERR, "TOO FEW ARGUMENTS\n");
        RETURN 1;
    }
    FD = OPEN (ARGV[1], O_RDONLY);
    IF (FD == -1) {
        FPRINTF (STDERR, "CANNOT OPEN FILE (%S)\n",
            ARGV[1]);
        RETURN 1;
    }
    WHILE ((BYTES = READ (FD, BUFFER, BUF_SIZE)) > 0)
    {
        BUFFER [BYTES] = 0;
        PRINTF ("%S", BUFFER);
    }
    CLOSE (FD);
    RETURN 0;
}

```

Бағдарламаның осы нұсқасында жаңа bytes айнымалысы пайда болды, онда оқылған ақпараттың мөлшері жазылады. Буфер-буфер массиві. Назар аударыңыз, әр уақытта нөлдік терминатор осы массивтің соңына қосылады. Бұл шығуды printf () функциясы жүзеге асыратындығына байланысты, ол үшін 0 коды бар таңба Шығыс аяқталуының белгісі ретінде қызмет етеді. Әрқашан есте сақтау керек, read () жүйелік қоңырауы байттарды оқиды, ал жолды қалыптастыру Жауапкершілігі бағдарламашыға жүктеледі.

Printf () функциясы деректерді терминал экранына шығаруға арналған. Сондықтан ол кейбір байттарды "сол күйінде" қабылдай алмайды. Келесі пәрменді қарастырыңыз:

```
$ ./read1 foo1 > foo2
```

Бұл жағдайда foo1 файлы foo2-ге көшіріледі, егер бастапқы файлда printf() пішімдеу командалары ретінде қабылдайтын белгілі бір таңбалар болмаса (қоңырау, 0 коды бар таңба және т.б.). Осыған байланысты read1 бағдарламасы файлдарды көшірудің әмбебап құралы бола алмайды.




```
#INCLUDE <STDIO.H>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#define BUF_SIZE 4096
char buffer [BUF_SIZE];
int main (int argc, char ** argv)
{
    int ifd, i;
    FILE * outfile;
    ssize_t bytes;
    if (argc < 3) {
        fprintf (stderr, "Too few arguments\n");
        return 0;
    }
    ifd = open (argv[1], O_RDONLY);
    if (ifd == -1) {
        fprintf (stderr, "Cannot open input file ""(%s)\n", argv[1]);
        return 1;
    }
    outfile = fopen (argv[2], "w");
    if (outfile == NULL) {
        fprintf (stderr, "Cannot open output file ""(%s)\n", argv[2]);
        return 1;
    }
    while ((bytes = read (ifd, buffer, BUF_SIZE)) > 0) {
        for (i = 0; i < bytes; i++) fputc (buffer[i], outfile);
    }
    close (ifd);
    fclose (outfile);
    return 0;
}
```

Бұл бағдарлама кез-келген таңбаларға бей-жай қарамайтын fputs () функциясын қолдана отырып шығарады. Бұл жағдайда деректерді жазу циклде символдық түрде жүзеге асырылады, сондықтан нөлдік Терминаторды орнатудың қажеті жоқ.



● Файлды жазу:

write()

Unistd.h тақырып файлында жарияланған write () жүйелік қоңырауы, файлдарды жазуға арналған:

```
ssize_t write (int FD, const void * BUFFER, size_t SIZE);
```

Бұл жүйелік қоңырау BUFFER буферінен байттардың мөлшерін FD дескрипторы бар файлға жазуға тырысады. Қайтару мәні-нақты жазылған байттардың саны. Қате болған жағдайда -1 қайтарылады.



```
#INCLUDE <STDIO.H>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#define BUF_SIZE 4096
char buffer [BUF_SIZE];
int main (int argc, char ** argv)
{
    int ifd, ofd;
    ssize_t bytes;
    if (argc < 3) {
        fprintf (stderr, "Too few arguments\n");
        return 0;
    }
    ifd = open (argv[1], O_RDONLY);
    if (ifd == -1) {
        fprintf (stderr, "Cannot open input file ""(%s)\n", argv[1]);
        return 1;
    }
    ofd = open (argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0640);
    if (ofd == -1) {
        fprintf (stderr, "Cannot open output file ""(%s)\n", argv[2]);
        return 1;
    }
    while ((bytes = read (ifd, buffer, BUF_SIZE)) > 0)
        write (ofd, buffer, bytes);
    close (ifd);
    close (ofd);
    return 0;
}
```

Мысалдан көрініп тұрғандай, write() жүйелік қоңырауы оның "әріптесі" read () - ге өте ұқсас. Енді осы жүйелік қоңыраулар арқылы консольді енгізу-шығару қалай жүзеге асырылатындығын қарастырайық.



ҚОРШАҒАН ОРТАНЫ ҚЫСҚАША ЕСКЕ ТҮСІРІҢІЗ. БІЗ ОНЫҢ АТА-АНАЛЫҚ ПРОЦЕСТЕН БАЛАҒА МҰРА БОЛАТЫНДЫҒЫН БІЛЕМІЗ. ФАЙЛДАРДА ДА ДӘЛ СОЛАЙ БОЛАДЫ: ПРОЦЕСС БАСТАЛҒАН КЕЗДЕ ДЕСКРИПТОРЛАР КЕСТЕСІ АТА-АНАНЫҢ АШЫҚ ФАЙЛДАРЫМЕН ТОЛТЫРЫЛАДЫ. БІРАҚ, ӘР ПРОЦЕСТЕ ТӘУЕЛСІЗ КӨШІРМЕМЕН ҰСЫНЫЛҒАН ОРТАДАН АЙЫРМАШЫЛЫҒЫ, ДЕСКРИПТОРЛАРДЫҢ МҰРАГЕРЛІГІНІҢ ЕКІ ЕРЕКШЕЛІГІ БАР: ФАЙЛ ОЛ АШЫЛҒАН БАРЛЫҚ ПРОЦЕСТЕР ЖАБЫЛҒАНША ЖАБЫЛМАЙДЫ. БАСҚАША АЙТҚАНДА, ЕГЕР АШЫҚ F ФАЙЛЫ БАР А ПРОЦЕСІ В ПРОЦЕСІН ТУДЫРСА, ОНДА F ФАЙЛЫН ЖАБУ ҮШІН ЕКІ ПРОЦЕСС (А ЖӘНЕ В) ОСЫ ФАЙЛДЫ ЖАБУЫ КЕРЕК. БҰЛ ШАРТ ЖЕТКІЛІКСІЗ ЕКЕНІН ЕСКЕРІҢІЗ, ӨЙТКЕНІ F ФАЙЛЫ А ПРОЦЕСІН АТА-АНАСЫНАН АЛУЫ МҮМКІН; ЕГЕР ПРОЦЕСТЕР ОРТАҚ ФАЙЛ ДЕСКРИПТОРЫН БӨЛІССЕ, ОНДА ОЛАР АҒЫМДАҒЫ ЕНГІЗУ-ШЫҒАРУ ПОЗИЦИЯСЫН ДА БӨЛІСЕДІ.

ӘДЕТТЕ, ЕГЕР ҚАРАПАЙЫМ ФАЙЛДАР ТУРАЛЫ АЙТАТЫН БОЛСАҚ, АТА-АНА ДЕСКРИПТОРЛАР КЕСТЕСІНДЕ НЕ БАР ЕКЕНДІГІ ТУРАЛЫ "ҮНДЕМЕЙДІ". БҰЛ ЖАҒДАЙДА САБАҚТАСТЫҚ ТЕК РЕСМИ ТҮРДЕ САҚТАЛАДЫ. БІРАҚ ЕГЕР БІЗ КОНСОЛЬДІ ЕНГІЗУ-ШЫҒАРУ ТУРАЛЫ АЙТАТЫН БОЛСАҚ, ОНДА МҰНДА ҰРПАҚТАРҒА ТІРКЕЛГЕН НӨМІРЛЕРІ БАР ҮШ ДЕСКРИПТОРДЫ АЛУҒА КЕПІЛДІК БЕРІЛЕДІ:

- 0-СТАНДАРТТЫ ЕНГІЗУ (STDIN);
- 1-СТАНДАРТТЫ ШЫҒЫС (STDOUT);
- 2-СТАНДАРТТЫ ҚАТЕ АҒЫНЫ (STDERR).





ОСЫЛАЙША, WRITE() ЖҮЙЕЛІК ҚОҢЫРАУ АРҚЫЛЫ КЕЙБІР
ХАБАРЛАМАНЫ ЭКРАНҒА ШЫҒАРА АЛАМЫЗ:

```
WRITE (1, MESSAGE, STRLEN (MESSAGE));
```

КЕЙДЕ ТІКЕЛей ДЕСКРИПТОР НӨМІРЛЕРІНІҢ ОРНЫНА UNISTD.H
ФАЙЛЫНДА ЖАРИЯЛАНҒАН СИМВОЛДЫҚ ТҰРАҚТЫЛАР ҚОЛДАНЫЛАДЫ:

```
#DEFINE STDIN_FILENO 0  
#DEFINE STDOUT_FILENO 1  
#DEFINE STDERR_FILENO 2
```

КӨПТЕГЕН БАҒДАРЛАМАШЫЛАР МҰНДАЙ ЖАРНАМАЛАРДЫ АРТЫҚ ДЕП
САНАЙДЫ ЖӘНЕ ӘДЕТТЕГІ САНДАРДЫ ҚОЛДАНУДЫ ЖАЛҒАСТЫРАДЫ.

Листинг 7.9. Программа reverse.c

```
#INCLUDE <FCNTL.H>
#define BUF_SIZE 4096
CHAR BUFFER [BUF_SIZE];
INT MAIN (VOID)
{
    INT I;
    FOR (I = 0; (READ (0, &BUFFER[I], 1) > 0) ||
           (I < BUF_SIZE); I++) {
        IF (BUFFER[I] == '\N') {
            I--;
            BREAK;
        }
    }
    FOR (; I >= 0; I--) WRITE (1, &BUFFER[I], 1);

    WRITE (1, "\N", 1);
    RETURN 0;
}
```

Бұл қызық нәрсе болды:

\$./REVERSE

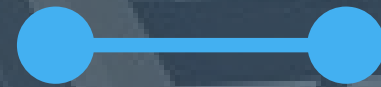
КАШАЛОТ КАК КОМПИЛЯТОР
РОТЯЛИПМОК КАК ТОЛАШАК

БІРАҚ БҰЛ БАҒДАРЛАМАНЫҢ
БАСТЫ ЕРЕКШЕЛІГІ-
КОНВЕЙЕРГЕ ҚАТЫСУ
МҮМКІНДІГІ:

\$ UNAME | ./REVERSE
XUNIL

Еркін қол жеткізу:

LSEEK()



LSEEK () ЖҮЙЕЛІК ҚОҢЫРАУЫ АҒЫМДАҒЫ
ЕНГІЗУ-ШЫҒАРУ ОРНЫН ЕРКІН ЖЫЛЖЫТУҒА
МҮМКІНДІК БЕРЕДІ. LSEEK () UNISTD.H
ТАҚЫРЫП ФАЙЛЫНДА КЕЛЕСІДЕЙ
ЖАРИЯЛАНАДЫ :

```
off_t lseek (int FD, off_t OFFSET, int WHENCE);
```

OFF_T ТҮРІ-ӨЛШЕМІ ІСКЕ АСЫРУҒА БАЙЛАНЫСТЫ
БҮТІН САН.

Әдетте, бұл ұзын бүтін (long int). FD-бұл файл
дескрипторы,

OFFSET-үш тұрақтының бірімен ұсынылуы мүмкін
WHENCE параметріне қатысты қажетті орын
ауыстыру:

SEEK_SET

файлдың басы;

SEEK_SET

ағымдағы енгізу-
шығару орны;

SEEK_END

файлдың соңы.



OFFSET АРГУМЕНТІ АҒЫМДАҒЫ ПОЗИЦИЯНЫ WHENCE-
ТЕН ЖЫЛЖЫТАТЫН ТЕРІС МӘНДЕРДІ ҚАБЫЛДАУЫ
МҮМКІН.

LSEEK () ЖҮЙЕЛІК ҚОҢЫРАУЫ ФАЙЛДЫҢ БАСТАЛУЫНА
ҚАРАЙ БЕЛГІЛЕНГЕН ЕНГІЗУ-ШЫҒАРУ ОРНЫН
ҚАЙТАРАДЫ. ҚАТЕ БОЛҒАН ЖАҒДАЙДА -1 ҚАЙТАРЫЛАДЫ.

LINUX ЯДРОСЫНДА ФАЙЛДАҒЫ АҒЫМДАҒЫ ҚОҢЫРАУ
ТУРАЛЫ ЕСЕП БЕРЕТІН ЖЕКЕ ЖҮЙЕЛІК ҚОҢЫРАУ ЖОҚ.
ОСЫ МАҚСАТТАР ҮШІН БІРДЕЙ LSEEK() ҚОЛДАНЫЛАДЫ:
OFF_T POS = LSEEK (FD, 0, SEEK_CUR);

Листинг 7.10. Программа mylseektail.c

```
#INCLUDE <FCNTL.H>
#include <UNISTD.H>
#include <SYS/TYPES.H>
#include <STRING.H>
#define BUF_SIZE      4096
CHAR BUFFER[BUF_SIZE];
INT MAIN (INT ARGV, CHAR ** ARGV)
{
    INT FD;
    OFF_T NBACK;
    SSIZE_T NBYTES;
    CHAR ARG_EMSG[] = "TOO FEW ARGUMENTS\n";
    CHAR FILE_EMSG[] = "CANNOT OPEN INPUT FILE\n";
    CHAR CLOSE_EMSG[] = "CANNOT CLOSE FILE\n";
    IF (ARGV < 3) {
        WRITE (2, ARG_EMSG, STRLEN (ARG_EMSG));
        RETURN 1;
    }
    FD = OPEN (ARGV[1], O_RDONLY);
    IF (FD == -1) {
        WRITE (2, FILE_EMSG, STRLEN (FILE_EMSG));
        RETURN 1;
    }
}
```

```
NBACK = ABS (atoi (argv[2]));
LSEEK (FD, 0, SEEK_END);
IF (NBACK > LSEEK (FD, 0, SEEK_CUR))
    LSEEK (FD, 0, SEEK_SET);
ELSE
    LSEEK (FD, -NBACK, SEEK_END);
WHILE ((NBYTES = READ (FD, BUFFER,
    BUF_SIZE)) > 0)
    WRITE (1, BUFFER, NBYTES);
IF (CLOSE (FD) == -1) {
    WRITE (2, CLOSE_EMSG, STRLEN
        (CLOSE_EMSG));
    RETURN 1;
}
RETURN 0;
```

Сонымен, 7.10 тізімін рет-ретімен қарастырайық. Fcntl.h тақырып файлында open () жүйелік қоңырауы жарияланды. Unistd.h файлы read(), write(), lseek() және close() үшін қажет. Sys/types-те.h түрлері off_t және size_t деп жарияланады, ал string.h бас файлы-strlen() функциясы үше қажет.

Содан кейін buf_size препроцессорының тұрақты мәні және деректер оқылатын және жазылатын буфер массиві жасалады. C тілінде статикалық массивтермен жұмыс істеудің тағы бір тәсілі бар. Ол үшін келесі макросты жариялау жеткілікті:

```
#define ARRAY_SIZE(array) (sizeof(array)/sizeof(array[0]))
```

Бұл макрос компиляция сатысында статикалық массив мөлшерін есептейді:

```
...
```

```
#define ARRAY_SIZE(array) (sizeof(array)/sizeof(array[0]))
```

```
char buffer[4096];
```

```
...
```

```
while ((nbytes = read (fd, buffer, ARRAY_SIZE (buffer)))
```

```
    write (1, buffer, nbytes);
```

```
...
```

Бұл бағдарламада барлық Енгізу-шығару операциялары қате туралы хабарламаларды шығаруды қоса алғанда, read() және write() жүйелік шақырулар көмегімен жүзеге асырылады. Ол үшін алдымен arg_msg, file_msg және close_msg массивтері жасалады.

arg_emsg, file_emsg және close_emsg массивтері статикалық екенін ескере отырып, strlen() функциясын мүлдем қолданбауға болады:

```
#define ARRAY_SIZE(array) (sizeof(array)/sizeof(array[0]))
```

```
write (2, arg_emsg, ARRAY_SIZE (arg_emsg));
```

```
write (2, file_emsg, ARRAY_SIZE (file_emsg));
```

```
write (2, close_emsg, ARRAY_SIZE (close_emsg));
```

ТАРАУДЫҢ
МАТЕРИАЛЫН БЕКІТУ
ҮШІН БІЗ КӨРСЕТІЛГЕН
ПОЗИЦИЯДАН
БЕРІЛГЕН БАЙТТАР
САНЫН ФАЙЛДАН
ОҚИТЫН ТАҒЫ БІР
БАҒДАРЛАМАНЫ
ҚАРАСТЫРАМЫЗ.
БАҒДАРЛАМАНЫ
"ЖЕТІЛДІРІЛГЕН" ЕТУ
ҮШІН ОНЫ
GETORT_LONG ()
ФУНКЦИЯСЫМЕН
ӨҢДЕУГЕ ҮЙРЕТЕМІЗ.
БАСТАУ ҮШІН БІЗ
ОПЦИЯЛАР МЕН
ДӘЛЕЛДЕР
ЖИЫНТЫҒЫН
АНЫҚТАЙМЫЗ.

1

-h (--help) опциясы

бағдарламамен жұмыс істеу туралы қысқаша анықтамалық ақпаратты көрсетеді.

2

-o (--output) опциясы

шығуды файлға бағыттау үшін қолданылады. Файл атауы тәуелді аргументте көрсетіледі. Егер -o опциясы көрсетілмесе, онда шығыс ақпарат стандартты қорытындыға жіберіледі.

3

-p (--position) опциясы

файлдағы позицияны бағдарламаға жібереді (тәуелді дәлел арқылы), онда ақпарат оқылады. Егер -p опциясы көрсетілмесе, онда Әдепкі бойынша оқу файлдың басынан басталады

4

-c (--count) опциясы

тәуелді дәлел арқылы бағдарламаға оқылуы керек байттар санын береді. Егер -h (--help) жалауы көрсетілмесе, бұл опция міндетті болып табылады.

5

-n (--newline) флагы

бағдарламаға нәтижені жолды беру арқылы аяқтау керектігін көрсетеді.

6

Міндетті тәуелсіз дәлел оқу орын алатын файл атауын бағдарламаға береді

Листинг 7.11. Пример реализации низкоуровневого ввода-вывода readblock.c

```
#INCLUDE <UNISTD.H>
#include <FCNTL.H>
#include <SYS/TYPES.H>
#include <SYS/STAT.H>
#include <GETOPT.H>
#include <STDLIB.H>
#define ARR_SIZE(ARRAY) (sizeof(ARRAY)/sizeof(ARRAY[0]))
int main (int argc, char ** argv) {
    int i, opt, ifd, ofd = 1, nflag = 0; char ch;
    off_t pos; size_t count;
    char help_str[] = "usage: readblock options filename\n" "options:\n" "-h, --help\n" "-o, --
output \n" "-p, --position \n" "-c, --count \n" "-n, --newline\n";
    char unkn_emsg[] = "unknown error\n";
    char ifile_emsg[] = "cannot open input file\n";
    char ofile_emsg[] = "cannot open output file\n";
    char close_emsg[] = "cannot close file\n";
    char lseek_emsg[] = "cannot set i/o position\n";
    char * ofname = NULL;
    char * pos_str = NULL;
    char * count_str = NULL;
    const struct option long_opts[] = {
        { "help", no_argument, NULL, 'h' },
        { "output", required_argument, NULL, 'o' },
        { "position", required_argument, NULL, 'p' },
        { "count", required_argument, NULL, 'c' },
        { "newline", no_argument, NULL, 'n' },
        { NULL, 0, NULL, 0 } };
    while ((opt = getopt_long (argc, argv, "ho:p:c:n",
                                long_opts, NULL)) != -1) {
        switch (opt) {
            case 'h':
                write (1, help_str, arr_size(help_str));
                return 0;
            case 'o':
                ofname = optarg;
                break;
            case 'p':
                pos_str = optarg;
                break;
            case 'c':
                count_str = optarg;
                break;
            case 'n':
                nflag = 1;
                break;
```

```
case '?':
    write (2, help_str, arr_size (help_str));
    return 1;
default:
    write (2, unkn_emsg, arr_size (unkn_emsg));
    return 2;
} }

if (count_str == NULL) {
    write (2, help_str, arr_size (help_str));
    return 3;
}
count = abs(atoi(count_str));
if (pos_str != NULL) pos = abs (atoi (pos_str));
else pos = 0;
if (optind >= argc) {
    write (2, help_str, arr_size (help_str));
    return 4;
}
if (ofname != NULL) {
    ofd = open (ofname, o_wronly | o_creat | o_trunc, s_irusr | s_iwusr | s_irgrp); /* 0640 */
if (ofd == -1) {
    write (2, ofile_emsg, arr_size (ofile_emsg)); return 5; }
}
ifd = open (argv[optind], o_rdonly);
if (ifd == -1) {
    write (2, ifile_emsg, arr_size (ifile_emsg));
    return 6;
}
if (pos > lseek (ifd, 0, seek_end)) {
    count = 0;
}
else if (lseek (ifd, pos, seek_set) == -1) {
    write (2, lseek_emsg, arr_size (lseek_emsg));
    return 7;
}
for (i = 0; i < count; i++) {
    if (read (ifd, &ch, 1) <= 0) break;
    write (ofd, &ch, 1);
}
if (nflag) write (ofd, "\n", 1);
if (close (ifd) == -1) {
    write (2, close_emsg, arr_size (close_emsg));
    return 8;
}
if (ofd != 1) {
if (close (ofd) == -1) {
    write (2, close_emsg, arr_size (close_emsg));
    return 9; }
}
return 0; }
```


Барлығын рет-ретімен қарастырайық. Алдымен бағдарламаға тақырып файлдары кіреді, олар мыналарды жариялайды:

- `unistd.h`-`read()`, `write()`, `lseek()` және `close ()` жүйелік қоңырауларды, сондай-ақ `seek_set`, `SEEK_CUR` және `SEEK_END` тұрақтыларын жариялайды;
- `fcntl.h`-`open ()` жүйелік қоңырауы, сондай-ақ `o_rdonly`, `O_WRONLY`, `O_CREAT` және `O_TRUNC` тұрақтылары;
- `sys/types.h`-түрлері `off_t` және `size_t`;
- `sys/stat.h`-`s_irsr`, `S_IWUSR` және `S_IRGRP` файл режимінің-тұрақтылары;
- `getopt.h`-`option` құрылымы, `optarg` және `optind` айнымалылары және `getopt_long()` функциясы;
- `stdlib.h`-`abs()` және `atoi ()` функциялары.

`ARR_SIZE ()` макросы құрастыру сатысында статикалық массив мөлшерін есептейді. `Main ()` функциясында алдымен әртүрлі қосалқы айнымалылар жарияланады:

- `i` бүтін айнымалы - цикл үшін есептегіш;
- `opt` бүтін айнымалы-`getopt_long ()` функциясы қайтаратын мәнді алу және өңдеу үшін;
- `ifd` бүтін айнымалы-кіріс файл дескрипторын сақтауға арналған;
- `ofd` бүтін айнымалы-шығыс файл дескрипторын сақтауға арналған, ол әдепкі бойынша (егер-о опциясы көрсетілмесе) стандартты шығумен байланысты;
- `nflag` бүтін айнымалы-нәтижені жолды беру арқылы аяқтау керектігін көрсетеді;
- `ch` символдық айнымалы — енгізу-шығару үшін;
- `pos` айнымалысы-файлдың бастапқы оқу орнын есептеу және сақтау үшін;
- `count` айнымалысы-шығарылатын байттар санын есептеу және сақтау үшін.

help_str, unkn_msg, ifile_emsg, ofile_emsg, close_emsg және lseek_emsg жолдары ақпараттық хабарламаларды шығаруға қызмет етеді. Әдетте, бұл қате туралы хабарламалар.

Ofname, pos_str және count_str көрсеткіштері сәйкесінше-o, -p және-C опцияларының тәуелді параметрлерін алу үшін қолданылады. Бастапқыда бұл көрсеткіштер null мәнімен инициализацияланған. Бұл белгілі бір мағынаға ие: егер, мысалы, параметрлерді өңдеу циклынан кейін ofname көрсеткіші әлі де NULL болса, онда бұл-o (--output) опциясы көрсетілмегенін білдіреді. long_opts құрылымдарының массиві ұзақ параметрлердің сипаттамасын және олардың дәлелдерін қамтиды. Сонымен қатар, бұл массив ұзақ және қысқа опциялар арасындағы сәйкестікті орнатады.

Мысалы, ұзақ опция-position қысқа -p опцияға сәйкес келеді.

Барлық хабарландырулардан кейін getopt_long() функциясы -1 мәнін қайтарғанға дейін жалғасатын опцияларды өңдеу циклі жүреді, бұл басқа өңдеуге ештеңе жоқ екенін білдіреді. Циклдің әр өтуімен getopt_long () функциясы алынған келесі опцияның орт айнымалы кодын қайтарады. Егер Белгісіз опция алынса, онда орт-ге таңба коды қойылады"?"(во просительный белгісі). Әр бағдарлама белгісіз опцияның пайда болуына басқаша жауап бере алады. Бұл жағдайда біздің бағдарлама стандартты ағынға шығады

қате туралы қысқаша анықтама (help_str) және 1 қайтару кодымен аяқталады. Егер getopt_long () басқа нәрсені қайтарады (switch дизайнының default блогы), қате туралы хабарлама пайда болады және бағдарлама 2 қайтару кодымен аяқталады.

Бұл бағдарламада әр қате бағдарламаны жеке қайтару кодымен аяқтайтынына назар аударыңыз. Бұл бағдарламада консольді енгізу-шығару `write()` жүйелік қоңырауы арқылы жүзеге асырылады. Осылайша, жеке енгізу-шығаруды жүзеге асыру үшін Сіз тек Linux ядросының көмегімен жасай аласыз. Бұл әрдайым ыңғайлы және орынды бола бермейді.

Бағдарламаның алдын-ала сипаттамасында бұл опция міндетті екендігі айтылды, сондықтан егер `count_str` айнымалы мәні әлі де `NULL` мәнін қамтыса, онда бағдарлама Қысқаша нұсқаулық жүргізеді және 3 қайтару кодымен аяқталады. Егер тексеру сәтті өтсе, онда `count_str` сандық мәнге аударылады, ол `count` айнымалысына енгізіледі. Бұл бағдарламада `abs()` модулін алу функциясы тексерулер санын азайтуға және бастапқы кодтың ықшамдылығын сақтауға арналған. Егер сіз `count` үшін теріс мәнді өңдеу блогын өзіңіз қоссаңыз жақсы болады.

Содан кейін `pos_str` өңделеді. Егер `-p` опциясы көрсетілсе, онда осы опцияның бос аргументі `abs()` және `atoi()` функцияларымен берілген `POS` айнымалысына бүкіл теріс емес мәнге қолданылады. Егер `pos_str` көрсеткіші `null` мәнін әлі де сақтаса (`-p` опциясы көрсетілмесе), онда `POS` айнымалысы нөлдік мәнге орнатылады. Бұл деректер файлдың басынан бастап оқылатындығын білдіреді.

Әрі қарай, кіріс файлының атауы енгізілетін міндетті бос Аргументтің болуын тексеру. Егер дәлел болмаса, онда стандартты қателер ағынына қысқаша анықтама (`help_str`) жіберіледі және Бағдарлама 4 қайтару кодымен аяқталады.

Осыдан кейін, егер `ofname` көрсеткішінің `null` мәнінен өзгеше болатын `-o` опциясы көрсетілсе, `O_WRONLY`, `O_CREAT` және `O_TRUNC` жалаушалары бар файлды `dos` типі 0640-тың сегіздік құқықтарына сәйкес режимде ашуға әрекет жасалады. Мұны `create()` жүйелік қоңырау арқылы жасауға болады. Егер файл ашылмаса, қате туралы хабарлама пайда болады және бағдарлама 5 қайтару кодымен аяқталады.

Содан кейін кіріс файлын `o_rdonly` жалаушасымен ашуға әрекет жасалады (тек оқу). Қате болған жағдайда қате туралы хабарлама шығады және бағдарлама 6 қайтару кодымен аяқталады.

Егер кіріс файлы сәтті ашылса, бағдарлама осы файл үшін оның басталуына қатысты ағымдағы енгізу-шығару орнын орнатуға көшеді. Бұл көрсетілген `pos` мәні файл өлшемінен асып кетуі мүмкін екенін ескереді. Бұл жағдайда `count` айнымалысы көрсетілген позициядан шығатын ештеңе жоқ екенін білдіретін сияқты қалпына келтіріледі. Егер `pos` айнымалысының мәні файл өлшемінен аспаса, онда ағымдағы енгізу-шығару орнын файлдағы басына қатысты көрсетілген байт санына орнатуға әрекет жасалады (`SEEK_SET`). Егер бірдеңе дұрыс болмаса, онда хабарлама (`help_str`) стандартты қателер ағынына шығады және Бағдарлама 7 қайтару кодымен аяқталады.

Ақырында, барлық алдын ала тексерулер сәтті өткен кезде, бағдарлама кіріс файлының таңбасының көрсетілген блогын таңба бойынша оқиды және әрбір таңбаны шығыс файлына жазады. Әрине, бұл ең жақсы нұсқа емес. Өзіңіз де солай істеуге тырысыңыз, бірақ буферлеу механизмін қолданыңыз.

Енді бәрі оқылып, жазылғаннан кейін файлдарды жабу керек. Енгізу файлын қандай да бір себептермен жабу мүмкін болмаса, қате туралы хабар көрсетіледі және бағдарлама қайтару коды 8 арқылы шығады. Шығыс файлды жабу үшін алдымен оның стандартты шығыс емес екеніне көз жеткізу керек. Егер шығыс файлын жабу кезінде қате орын алса, сәйкес хабарлама көрсетіледі және бағдарлама 9 қайтару кодымен шығады. Файлды шынымен жабу үшін, файлды қолданатын әрбір процесте `close()` шақырылуы керек.



НАЗАРЛАРЫҢЫЗҒА РАХМЕТ!

