ТЕМА 4. ВЗАИМОДЕЙСТВИЕ НА ПРОЦЕСИ ПОСРЕДСТВОМ КАНАЛИ

Понятие за входно-изходен поток. Работа с файлове, чрез системните извиквания и стандартната библиотека за вход-изход. Понятие за файлов дескриптор. Отваряне на файлове. Системно извикване open(). Системни извиквания close(), read() и write(). Понятие за неименуван канал (pipe). Системно извикване pipe(). Организиране на връзка, чрез неименуван канал между процес-родител и процес-дете. Наследяване на файловите дескриптори при извиквания fork() и exec(). Особености свързани с извикванията read() и write() при използване на pipe. Понятие за именуван канал (FIFO). Използване на системното извикване mknod() за създаване на именуван канал. Функция mkfifo(). Особености на извикването open() при отваряне на FIFO.

1. Комуникация между процеси. (или Междупроцесна комуникация).

Взаимодействието между процеси изисква специални средства за комуникация, които осигуряват предаването и получаването на данни (съобщения) между тях.

Принципно съществуват две схеми за комуникация – използване на обща памет и системи със съобщения.

При общата памет комуникиращите процеси обменят данни чрез общи променливи, а при системи със съобщения процесите могат да обменят съобщения, без да е необходимо да се обръщат към общи променливи.

Двата начина не се изключват взаимно в една ОС, те могат да се прилагат едновременно. Размяната на съобщения има две ограничения:

• подателят не може да надхвърли крайния обем на буфера за съобщения;

• получателят не може да обработва съобщения по-бързо, отколкото те се създават от процеса – подател.

Тези ограничения на ресурсите се изпълняват, ако се спазват

правилата за синхронизация:

• ако подател се опита да запише съобщение в пълен буфер, той се задържа до прочитане на буфера;

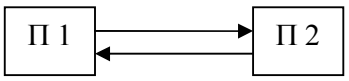
• ако получател се опита да вземе съобщение от празен буфер, той се задържа до получаване на съобщението.

Буферът обикновено се разполага в оперативната памет. Ако се използва външна памет, първото ограничение може да се снеме.

Системите за съобщения се реализират най-често чрез директна или индиректна комуникация. При директната комуникация (фиг. 1) всеки процес, изпращащ или получаващ съобщение, трябва явно да посочи името на процеса получател или подател. Характерно е, че установеният комуникационен канал е двупосочен, свързан само с двата процеса, и е единствен за тях. Той се установява автоматично, а процесите трябва да знаят само имената си.

При индиректната комуникация (фиг. 2) се използва буферна памет, наречена пощенска кутия, за съхраняване на изпратените, но неполучени съобщения. Всяка от тях си има уникално име като процесът комуникира с други процеси само с пощенски кутии.

Комуникационният канал може да бъде еднопосочен, двупосочен или свързан с повече от два процеса. Допуска се процес да създаде сам пощенска кутия и в този случай с унищожаване на процеса се унищожава и пощенската кутия.



Фиг. 1. Директна комуникация

Фиг. 2. Индиректна комуникация

От друга страна, когато пощенската кутия принадлежи към някакво множество, управлявано от ОС, тя е независима от процесите и затова съобщението трябва да съдържа имената на подателя и получателя. В този случай пощенската кутия е независима от процесите. Обикновено ОС

предоставя възможност на процесите да създават и да унищожават пощенски кутии. В най-простия случай комуникацията е еднопосочна, когато много процеси искат да общуват с един процес, например с важна управляваща програма (за вход или изход и т. н.) В много случаи е необходима двупосочна връзка, например получателят задължително изпраща потвърждение за полученото съобщение. За целта може да се използва пощенска кутия, свързана с два процеса и разрешаваща предаване на съобщение в двете посоки.

Каналите за връзка са най-често употребяваните средства за комуникация между процесите. Те осигуряват еднопосочно предаване на неформатиран поток от данни (поток от байтове) между процесите и синхронизация на работата им. Операциите по предаване и приемане на данните не зависят от съдържанието на това, което се предава или приема. Цялата информация в канала за връзка се разглежда като непрекъснат поток от байтове без да се взема предвид нейната вътрешна структура.

Реализират се два типа канали в различните версии на UNIX и Linux системите:

• неименуван канал (pipe) - за комуникация между родствени процеси

• именуван канал (named pipe или FIFO файл) - за комуникация между независими процеси.

Като механизъм за комуникация те са еднакви. Реализират се като тип файл, който се различава от обикновените файлове и има следните особености:

- За четене и писане в него се използват системните извиквания read и write, но дисциплината е FIFO.

- Каналът има ограничен капацитет (максималния брой байтове при една операция запис в канал при всички версии на UNIX не надхвърля няколко килобайта);

- Прочетената от канала информация незабавно се изтрива и неможе да се прочете повторно.

Двата типа канала се различават по начина, по който се създават и унищожават и по начина, по който процес осъществява първоначално достъп до канала.

2. Неименуван канал (pipe)

Неименуваният канал (pipe) е част от потоковия модел на операционната система UNIX, който позволява директно предаване на информация (директна комуникация) между процеси в операционната система.

Неименуваният канал се разполага в адресното пространство на ОС. Достъпът до входа и изхода на канала се осъществява с помощта на системните извиквания *read()* и *write()*. На практика, *pipe* представлява обособена област от паметта, организирана във вид на кръгов буфер. При четене и запис в буфера се преместват два указателя, които съответстват на входните и изходни потоци. При това изходния указател не може да припокрива входния и обратно.

За създаване на нов неименуван канал (нов кръгов буфер вътре в операционната система) се използва системното извикване *pipe()*.

ИМЕ

pipe

ПРОТОТИП

```
#include <unistd.h>
int pipe(int *fd);
```

ОПИСАНИЕ

Системното извикване pipe е предназначено за създаване на канал за директна комуникация между процеси в операционната система.
Параметърът fd е указател към масив от две целочислени променливи. При нормално завършване на извикването в първия елемент на масива – fd[0] – се записва файловият дескриптор, който съответства на изходния поток от данни и позволява операцията четене, а във вторият елемент на масива – fd[1] – се записва файловият дескриптор, който съответства на входния поток от данни и позволява изпълнение на операцията запис.

ВРЪЩАНИ СТОЙНОСТИ

Системното извикване връща стойност 0 при нормално завършване и стойност -1 при възникване на грешка.

В процеса на работа, системното извикване заделя област от паметта под формата на буфер и указатели. В буфера се съхранява и чете информацията, съответстваща на входния и изходен поток от данни. В два елемента от таблицата на отворените файлове се записва информацията, съответстваща на двата потока (входен и изходен). По този начин всеки неименуван канал се асоциира с два файлови дескриптора. За първия е разрешена операцията четене, а за втория – операцията запис в канала. За изпълнение на тези операции се използват системните извиквания *read()* и

write(), по същия начин, както при работа с файлове. При завършване на използването на входно-изходния поток е необходимо, той да се затвори с помощта на системното извикване *close()*. Така се освобождават заетите системни ресурси. Когато всички процеси, използващи неименувани канали, затворят асоциираните с тях файлови дескриптори, операционната система изтрива *pipe* от паметта. Така времето за съществуване на неименуваните каналите в системата не може да превиши времето на жизния цикъл на процесите, работещи с тях.

Представената програма 4_2.c илюстрира работата с *pipe* в рамките на един процес – създаване на канала, запис и четене, и освобождаване на заделените ресурси.

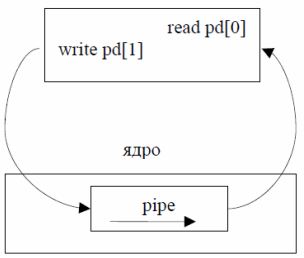
```
/* Програма 4_2.c, иллюстрирюща работата с pipe в рамките на
  един процес */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(){
    int fd[2];
    size_t size;
    char string[] = "Hello, world!";
    char resstring[14];
    /* Опит да се създаде pipe */
    if(pipe(fd) < 0){
        /* Ако създаването на pipe е неуспешно се отпечатва
        съобщиние за грешка и се прекратява работата */
        printf("Can\'t create pipe\n");
        exit(-1);
    }
    /* В създадения канал се записват 14 байта от символен масив,
    т.е. целия низ "Hello, world!" заедно с знака за край на низа */
    size = write(fd[1], string, 14);
    if(size != 14){
        /* Ако са записани повече на брой байтове се съобщава за
        грешка */
        printf("Can\'t write all string\n");
        exit(-1);
    }
    /* От канала се прочитат 14 байта в друг масив, т.е.
    целия записан низ */
    size = read(fd[0], resstring, 14);
    if(size < 0){
        /* Ако четенето е неуспешно се съобщава за грешка */
        printf("Can\'t read string\n");
        exit(-1);
    }
    /* Прочетения низ се отпечатва */
    printf("%s\n",resstring);
    /* Входния поток се затваря*/
    if(close(fd[0]) < 0){
        printf("Can\'t close input stream\n");
    }
    /* Изходния поток се затваря */
    if(close(fd[1]) < 0){
        printf("Can\'t close output stream\n");
    }
    return 0;
}
```

Програма 4_2.c. Програма, илюстрираща работата на неименуван канал в рамките на

един процес

Задача 3: Въведете програмата, компилирайте и я изпълнете.

На фиг. 3 е представен достъпа до неименуван канал в рамките на един процес.



Фиг. 3. Неименуван канал в рамките на един процес

3. Свързване на близко родствени процеси посредством неименуван канал

Неименуваните канали заменят функцията по копиране на памет в памет вътре в процеса с предаване на информацията, чрез операционната система. При пораждане на нов процес със системното извикване *fork()* процеса-дете наследява таблицата на отворените файлове от процеса-родител. Тази таблица остава неизменна част от системния контекст на процеса и при използване на системното извикване *exec()*. Това позволява дасе използва pipe, за да се организира комуникация между процеси с общ прародител (създател на неименувания канал).

В програмата 4_3.c се реализира еднопосочна връзка между процес-родител и процес-дете.

```
/* Програма 4_3.c, осъществяваща еднопосочна връзка чрез неименуван
   канал между процес-родител и процес-дете */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(){
    int fd[2], result;
    size_t size;
    char resstring[14];
    /* Създаване на неименувания канал */
    if(pipe(fd) < 0){
        /* Ако създаването на pipe е неуспешно се отпечатва съобщение
        и се прекратявя работата */
        printf("Can\'t create pipe\n");
        exit(-1);
    }
    /* Пораждане на нов процес */
```

```
    result = fork();
    if(result){
        /* Ако създаването на процеса е неуспешно се съобщава за това и
        се прекратява програмата */
        printf("Can\'t fork child\n");
        exit(-1);
    } else if (result > 0) {
        /* Родителския процес, передава информация към процеса-дете.
        Изходния поток не му е необходим, затова се затваря */
        close(fd[0]);
        /* В канала се записват 14 байта, т.е. низа
        "Hello, world!" заедно със знака за край на низ */
        size = write(fd[1], "Hello, world!", 14);
        if(size != 14){
            /* Ако са записани по-голям брой байтове, се съобщава
            за грешка и програмата се прекратява */
            printf("Can\'t write all string\n");
            exit(-1);
        }
        /* Входния поток от данни се затваря и се прекратява
        работата на процеса-родител */
        close(fd[1]);
        printf("Parent exit\n");
    } else {
        /* Породеният процес получава информация от
        процеса-родител. Той унаследява от него
        таблицата на отворените файлове. По този начин, той
        може да използва файловите дескриптори, съответстващи
        на канала. В този процес входният поток от данни не е
        необходим, затова се затваря.*/
        close(fd[1]);
        /* От канала се прочита в 14 байтов в масив
        целия записан низ */
        size = read(fd[0], resstring, 14);
        if(size < 0){
            /* Ако четенето е неуспешно се съобщава за грешка
            и програмата се прекратява */
            printf("Can\'t read string\n");
            exit(-1);
        }
        /* Прочетения низ се отпечатва */
        printf("%s\n",resstring);
        /* Входния поток се затваря и програмата завършва
           изпълнението си */
        close(fd[0]);
    }
    return 0;
}
```

Програма 4_3.c. Програма, осъществяваща еднопосочна връзка, чрез pipe между процес-родител и процес-дете

Задача 4: Въведете програмата, компилирайте и я изпълнете.

Задача 5 (повишена сложност): Промнете програмата 4_3.c, така че да свържете два родствени процеса, изпълняващи различни програми.

Неименуваният канал служи за организация на еднопосочна или симплексна връзка. При организиране на двупосочна връзка, чрез *pipe,* когато процеса-родител записва информация в канала се предполага, че нейния получател е процеса-дете. Също така се предполага, че прочетената информация от процеса-родител е записана от породения процес.

Възможно е да възникне ситуация, при която процеса-родител да прочете информацията, която самият той е записал, а процеса-дете да не получи нищо. Ако се използва един канал за двупосочно предаване на информация са необходими специални средства за синхронизация между процесите. Един от начинте да се организира двупосочна връзка между родствени процеси е чрез използване на два канала.

Задача 6: Променете програмата от листинг 4.3, за да организирате двупосочна връзка между родствени порцеси. Компилирйте програмата и я изпълнете.

4. Особености свързани с извикванията read() и write() при използване на pipe

Системните извиквания *read()* и *write()* имат определени особености в своето поведене при работа с *pipe*. Тези особености са свързани с ограничения размер на този вид канал, задръжката, която има при передаване на данните и възможността за блокиране на процесите, обменящи си информация по канала.

Необходимо е да се съобразява обема на информацията предавана, чрез *pipe*. Тъй като неименувания канал има ограничен капацитет, не са рядко случаите, когато от *pipe* се прочита по-малко информация, отколкото е била изпратена, а и при запис на информация в канала може да се запише по-малко от планираното. Това налага при създаването на програми с използване на канали да се проверяват стойностите връщани от системните извиквания.

Една от особеностите в поведението на системното извикване *read()* е свързана с блокиране при опит за чете от празен *pipe*. Ако съществуват процеси, в които *pipe* е отворен за запис, то системното извикване *read()* блокира до поява (запис) на информация в канала. Ако няма такива процеси системното изивикване *read()* връща стойност *0* без да блокира процеса. Тази особеност води до необходимостта от затваряне на файловия дескриптор, свързан с входния поток на канала, в процеса, използващ *pipe* за четене (*close(fd[1])* в процеса-дете в програмата от листинг 4.3). Аналогично е поведението на системното извикване *write()* при отсъствие на процеси, в които *pipe* е отворен за четене, от което следва необходимостта за затваряне на файловия дескриптор, свързан с изходния

поток на канала, в процеса, използващ *pipe* за запис (*close(fd[0])* в процеса-родител в програмата от листинг 4.3).

ИМЕ

read

ОСОБЕНОСТИ ПРИ РАБОТА С PIPE, FIFO И SOCKET

Ситуация	*Поведение*
Опит да бъдат прочетеneи по-малко байтове от наличните в канала за връзка.	Прочита се неободимото количество байтове и се връща стойност, съответстваща на прочетеното количество. Прочетената информация се изтрива от канала за връзка.
В канала за връзка се намират по-мако от необходимите байтове, но повече от нула.	Прочита се всичко налично в канала и се връща стойност, съответстваща на прочетеното количество. Прочетената информация се изтрива от канала за връзка.
Опит да се чете от канал за връзка, в който няма налична информация. Блокирането на извикването е разрешено.	Извикването блокира до момента, в който се появи информация в канала за връзка и до съществуването на процес, който може да предава информация в този канал. Ако се появи информация, процеса се разблокира, и поведението на извикването съответства на описаното в двата предходни реда от таблицата. Ако в канала никой не предава данни (няма нито един процес, за който този канал за връзка е отворен за запис), то извикването връща стойност 0. Ако канала за връзка бъде напълно затворен за запис във времето, когато четящия процес е блокиран, то той се разблокира и системното извикване връща стойност 0.
Опит да се чете от канал за връзка, в който няма налична информация. Блокирането на извикването не е разрешено.	Ако има процеси, за които канала за връзка е отворен за запис, системното извикване връща стойност -1 и записва стойност EAGAIN в променливата errno. Ако такива процеси липсват, системното извикване връща стойност 0.

ИМЕ

write

ОСОБЕНОСТИ ПРИ РАБОТА С PIPE, FIFO И SOCKET

Ситуация	*Поведение*
Опит да се запишат в канала за връзка по-малко байтове, отколкото остават до неговото запълване.	Необходимото количество байтове се помества в канала за врзъка, връща се записаното количество байтове.
Опит да се запишат в канала за връзка повече байтове, отколкото остават до неговото запълване. Блокирането на извикването е разрешено.	Извикването блокира, до момента, когато всички данни бъдат поместени в канала за връзка. Ако размера на буфера на канала за връзка е по-малък, от предаваното количество информация, то извикването, само започва да чака, докато част от информацията не бъде прочетена от канала за връзка. Връща записаното количество байтове.
Опит да се запишат в канала за връзка повече байтове, отколкото остават до неговото запълване, но по-малко, от размера на буфера на канала за връзка. Блокирането на извикването не е разрешено.	Системното извикване връща стойност -1 и записва стойност EAGAIN в променливата errno.
Канала за връзка не е запълнен. Опит да се запишат в него повече байтове,	Записват се толкова байта, колкото остават до запълване на канала. Системното извикване

отколкото остават до запълването му и повече от размера на неговия буфер. Блокирането на извикването не е разрешено.	връща записаното количеството байтове.
Опит да се записва в канал за връзка, в който няма място. Блокирането на извикването не е разрешено.	Системното извикване връща стойност -1 и записва стойност EAGAIN в променливата errno.
Опит да се записва в канал за връзка, от който никой няма повече да чете, или канлът напълно се затваря за четене по време на блокировката на системното извикване.	Ако изивкването е блокирано, то се разблокира. Процесът получава сигнал SIGPIPE. Ако този сигнал се обработва от потребителя, то системното извикване връща стойност -1 и записва стойност EPIPE в променливата errno.

Необходимо е да се отбележи и следната допълнителна особеност за системното изивкване *write* при работа с pipe и FIFO. Записът на информация, размерът на която не превишава размера на буфера, трябва да се осъществява атомарно – в рамкита на една инструкция. Това обяснява някой блокировки и грешки в дадения по-горе списък.

Задача 7 (повишена сложност): Определете размера на *pipe* за вашата операционна система.

5. Именуван канал (FIFO)

Както бе изяснено, достъпа до информацията за разположението на *pipe* в операционната система и неговото състояние може да бъде осъществен единствено, чрез таблицата на отворените файлове на процеса, създал канала и чрез унаследените от него таблици на отворените файлове на процесите-деца. Поради тази причина изложеният по-горе механизъм за обмен на информация, чрез *pipe* е възможен или за родствени процеси, имащи общ прародител, инициирал системното извикване *pipe()*, или за тези процеси и самият прародител и не може да се използва за потоков обмен с други процеси. В операционната система UNIX съществува възможността *pipe* да се използва и за взаимодействие между неродствени процеси, но нейната реализация е достатъчно сложна за да бъде разгледана в рамките на настоящото упражнение.

За организиране на потоково взаимодействие на какви да е процеси в операционната система UNIX се използва средство за връзка, получило названието FIFO (от First Input First Output) или именуван канал. Именуваният канал е подобен на неименувания (*pipe*) с едно изключение: данните за разположението на FIFO се намират в адресното пространство на ядрото на операционната система и процесите могат да получат достъп до него не чрез родствените си връзки, а чрез файловата система. За това при създаване на именуван канал на диска се създава файл от специален тип, към който процесите могат да се обръщат, за да получават

интересуващата ги информация. За създаването на FIFO се използва системното извикване *mknod()* или съществуващата в някои версии на UNIX функция *mkfifo()*.

Следва да се отбележи, че от тяхната работа не произтича действително заделяне на области от адресното пространство на операционната система за именувани канали, а само се създава файл, съществуването, на който позволява реално да се осъществи FIFO организация в паметта, след като този файл бъде отворен с помоща на системно изивикване *open()*.

След отваряне на именувания канал, с него се работи, точно както и с неименувания. За последващата работа с него се използват системните извиквания *read()*, *write()* и *close()*. Времето за съществуването на FIFO в адресното пространство на ядрото на операционната система, както и в случая с *pipe*, не може да превишава времето на живот на последният от използващите го процеси. Когато всички процеси, работещи с FIFO, затворят всички файлови дескриптори, асоциирани с него, системата освобождава ресурсите, заделени за неговото организиране. Непрочетената информация се губи. В същото време файлът създаден за именуван канал остава на диска и може да се използва за ново организиране на FIFO в бъдеще.

ИМЕ

mknod

ПРОТОТИП

```
#include <sys/stat.h>
#include <unistd.h>
int mknod(char *path, int mode, int dev);
```

ОПИСАНИЕ

Тук не е направено пълно описание на системното извикване mknod, а само описание на неговото използване за създаване на FIFO, като са разгледани само онези негови параметри, които засягат тази специфична дейност.
Параметърът dev е несъществен в тази ситуация и винаги може да бъде равен на 0.
Параметърът path е указател към низ, съдържащ пълното или относително име на файла, който се използва като етикет при създаването на FIFO. Не е задължително файл с такова име предварително да съществува на диска.
Параметърът mode задава атрибутите за правата за достъп на различните категории потребители към FIFO. Този параметър се задава като резултат от побитовата операция "или" между стойноста S_IFIFO, указваща, че системното извикване трябва да създаде FIFO и някаква сума от следните осмични стойности:

- 0400 – разрешено четенето за потребителя, създаващ FIFO;
- 0200 – разрешен записа за потребителя, създаващ FIFO;
- 0040 – разрешено четенето за групата от потребители, създаващи FIFO;
- 0020 – разрешен записа за групата от потребители, създаващи FIFO;

• 0004 – разрешено четенето за всички останали потребители;
• 0002 – разрешен записа за всички останали потребители.
При създаване на FIFO реално установените права за достъп се получават като комбинация от параметра mode и маската, използвана за създаване на файлове от текущия процес umask, а именно – (0777 & mode) & ~umask.

ВРЪЩАНИ СТОЙНОСТИ

При успешно създаване на FIFO системното извикване връща стойност 0, при неуспешно – отрицателна стойност.

ИМЕ

mkfifo

ПРОТОТИП

```
#include <sys/stat.h>
#include <unistd.h>
int mkfifo(char *path, int mode);
```

ОПИСАНИЕ

Функцията mkfifo е предназначена за създаване на FIFO в операционната система.
Параметърът path е указател към низ, съдържащ пълното или относително име на файла, който се използва като етикет при създаването на FIFO. Не е задължително файл с такова име предварително да съществува на диска.
Параметърът mode задава атрибутите за правата за достъп на различните категории потребители към FIFO. Този параметър се задава като някаква сума от следните осмични стойности:
• 0400 – разрешено четенето за потребителя, създаващ FIFO;
• 0200 – разрешен записа за потребителя, създаващ FIFO;
• 0040 – разрешено четенето за групата от потребители, създаващи FIFO;
• 0020 – разрешен записа за групата от потребители, създаващи FIFO;
• 0004 – разрешено четенето за всички останали потребители;
• 0002 – разрешен записа за всички останали потребители.
При създаване на FIFO реално установените права за достъп се получават като комбинация от параметра mode и маската, използвана за създаване на файлове от текущия процес umask, а именно – (0777 & mode) & ~umask.

ВРЪЩАНИ СТОЙНОСТИ

При успешно създаване на FIFO системното извикване връща стойност 0, при неуспешно – отрицателна стойност.

Важно е да се разбере, че файлът от тип FIFO не служи за разполагане на информацията записана в именувания канал на диска. Тази информация се разполага вътре в адресното пространство на операционната система, а файлът се явява само етикет, създаващ предпоставки за нейното разположение.

6. Особености в поведението на системното извикване open() при отваряне на FIFO

При работа с FIFO системните извиквания *read()* и *write()* имат същите особености, както и при работа с *pipe*. Системното извикване *open()* има следните особености при работа с FIFO: 1. Ако FIFO се отваря само за четене и не е установен флага O_NDELAY, то процеса, осъществяващ системното извикване, блокира до момента, в който друг процес не отвори

FIFO за запис. При това, ако флагът O_NDELAY е установен, то като стойност се връща асоциирания с FIFO файлов дескриптор; 2. Ако FIFO е отворен само за запис и флагът O_NDELAY не е установен, то процеса, осъществяващ системното извикване блокира до монента, в който друг процес не отвори FIFO за четене. Ако флагът O_NDELAY е установен, то се констатира възникване на грешка и се връща стойност *-1*; 3. Ако флага O_NDELAY е установен в параметрите на системното извикване *open()*, то процеса, отварящ FIFO не блокира при изпълнение на последващата операция с канала (четене или запис).

За илюстрация на взаимодействието между процеси чрез FIFO е дадена следната програма:

```
/* Програма 4_4.c, осъществяваща еднопосочна връзка, чрез FIFO между
   процес-родител и процес-дете */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
int main(){
    int fd, result;
    size_t size;
    char resstring[14];
    char name[]="aaa.fifo";
    /* Нулиране на маската, използвана за създаване на файлове от
       текущия процес, така че правата за достъп за създадения FIFO
       точно  да съответстват на параметъра на изивкването mknod() */
    (void)umask(0);
    /* Опит за създаване на FIFO с име aaa.fifo в текущата
       директория */
    if(mknod(name, S_IFIFO | 0666, 0) < 0){
        /* Ако създаването на FIFO не успее се отпечатва
           съобщение за това и работата се прекратява */
        printf("Can\'t create FIFO\n");
        exit(-1);
    }
    /* Поражда се нов процес */
    if((result = fork()) < 0){
        /* Ако създаването на процеса не успее се отпечатва
           съобщение за това и работата се прекратява */
        printf("Can\'t fork child\n");
        exit(-1);
    } else if (result > 0) {
        /* В родителския процес, който предава информацията към
           процеса-дете се отваря FIFO за запис.*/
        if((fd = open(name, O_WRONLY)) < 0){
            /* Ако отварянето на FIFO не успее се отпечатва
           съобщение за това и работата се прекратява */
            printf("Can\'t open FIFO for writing\n");
            exit(-1);
        }
        /* Опит за запис във FIFO на 14 байта, т.е. целия низ
        "Hello, world!" заедно със знака за край на низ */
        size = write(fd, "Hello, world!", 14);
        if(size != 14){
        /* Ако бъдат записани по-малък брой байтове, то се
           съобщава за грешка и работата се прекратява */
            printf("Can\'t write all string to FIFO\n");
            exit(-1);
        }
        /* Затваря се входния поток за данни и с това родителя
```

```
            приключва работата си */
        close(fd);
        printf("Parent exit\n");
    } else {
        /* В породения процес, който получава информация от
           процеса-родител се отваря FIFO за четене.*/
        if((fd = open(name, O_RDONLY)) < 0){
            /* Ако отварянето на FIFO не успее се отпечатва
           съобщение за това и работата се прекратява */
            printf("Can\'t open FIFO for reading\n");
            exit(-1);
        }
        /* Опит за четене от FIFO на 14 байта в масив, т.е.
           целия записан низ */
        size = read(fd, resstring, 14);
        if(size < 0){
            /* Ако четенето не успее се съобщава за грешка
               и работата завършва */
            printf("Can\'t read string\n");
            exit(-1);
        }
        /* Отпечатва се прочетения низ */
        printf("%s\n",resstring);
        /* Затваря се входния поток и работата завършва */
        close(fd);
    }
    return 0;
}
```

Листинг 4.4. Програма 4_4.c, осъществяваща еднопосочна връзка, чрез FIFO между процес-родител и процес-дете

Задача 7: Въведете програмата, компилирайте и я изпълнете.

Съвет: Обърнете внимание, че повторното стартиране на тази програма води до грешка при опит да бъде създаден FIFO, тъй като файлът със зададеното име, вече съществува. Ето защо е необходимо този файл да бъде или ръчно изтрит от диска, преди повторното стартране на програмата, или от изходния код на програмата да бъде изключено всичко свързано със системното извикване *mknod()*.

Задача 8: Базирайкки се на предходия пример, напишете две програми, едната от които записва информация във FIFO, а втората – чете от него, така че между тях да липсва ярко изразена родствена връзка (т.е. нито една от тях да не е наследник на другата).

Пояснение: Ако програмите от предходната точка бъдат стартирани на два отделни компютъра, които имат споделена файлова система (например, такава монтирана с помоща на NFS), така че всяка програма да работи на свой компютър, а FIFO се създаде на споделената файлова система. Въпреки, че двата процеса, ще имат достъп и ще използват един и същ файл от тип FIFO, взаимодействието между тях няма да се осъществи, тъй като те ще работят във

физически различни адресни пространства и ще се опитат да отворят FIFO в рамките на различни операционни системи.