ТЕМА 3. СИГНАЛИ В ОПЕРАЦИОННАТА СИСТЕМА UNIX

Понятие за сигнал. Видове сигнали, според източника и начини за тяхната обработка. Понятие за групи от процеси, сесия, лидер на групата, лидер на сесията, управляващ терминал на сесията. Системни извиквания getpgrp(), setpgrp(), getpgid(), setpgid(), getsid(), setsid(). Системно извикване kill и команда kill(). Системно извикване signal(). Създаване на потребителски функции за обработка на сигнали. Възстановяване на предходната реакция на сигнал. Сигнали SIGUSR1 и SIGUSR2. Използване на сигнали за синхронизация на процеси. Завършване на породен процес. Системно извикване waitpid(). Сигнал SIGCHLD. Понятие за надеждност на сигналите. POSIX-функции за работа със сигнали.

1. Видове сигнали

Един от начините за управление на процеси (включително комуникация между процеси) в UNIX е посредством сигнали. Използват се във всички версии на UNIX и Linux, заради своята ефективност и относително проста реализация.

Сигналите са кратки съобщения, които ядрото изпраща към процес или група от процеси. Те информират процеса, че е настъпило някакво събитие в ОС. Всеки сигнал притежава уникален номер в диапазона от 1 до 31 включително (табл. 1). Това е и единствената информация, която се съдържа в един сигнал, т.е процеса получава само номера, идентифициращ сигнала и никаква друга придружаваща информация.

Сигналите могат да се класифицират в зависимост от събитието, предизивкало изпращането на сигнал:

1) Сигнали свързани с апаратно особени ситуации. Сигналите в тази категория са свързани със събития, откривани от апаратурата и сигнализирани чрез прекъсване. Ядрото реагира на това като изпраща сигнал на процеса, който в момента се намира в състояние "Изпълнение в потребителски режим".

2) Сигнали изпращани по заявка от процеси. Ядрото реагира на заявка от процес, изпълняващ системно извикване *kill()* и предава заявения

сигнал към определен процес или група от процеси.

3) Сигнали свързани с програмно обособени ситуации. Сигналите в тази категория възникват в операционната система по повод различни синхронни или асинхронни събития свързани с процеса, на който са изпратени. Те имат чисто програмен характер и не са предизивкани от апаратурата.

4) Сигнали, свързани с управляващия терминал. Тези сигнали се изпращат от ядрото на процес или група от процеси при натискане на определена клавишна комбинация.

5) Сигнали свързани със системата за управление на заданията. Изпращат се при изпълнение на командата *kill*.

Ситуациите в които изпращането на сигнал към процес или група от процеси е инициирано от друг процес (това са сигналите от категории 2, 3 и 5), могат да се разглеждат като взаимодействия на процеси посредством сигнали.

Таблица 1. Описание на част от сигналите в UNIX

№	Име	Реакция по подразбиане	Събитие
1	SIGHUP	Безусловно завършване на процеса	Прекъсване на връзката с управляващия терминал
2	SIGINT	Безусловно завършване на процеса	Хардуерно прекъсване от калвиатурата – натискане на клавишиа <Del> или <Ctrl>+<c>
3	SIGQUIT	Завършване на процеса със създаване на core файл	Хардуерно прекъсване от калвиатурата – натискане на клавишите <Ctrl>+</>
4	SIGILL	Завършване на процеса със създаване на core файл	Опит за изпълнение на недопустима инструкция
8	SIGFPE	Завършване на процеса със създаване на core файл	Деление на 0 или препълване при операции с плаваща запетая
9	SIGKILL	Безусловно завършване на процеса	Предизвикано завършване на процеса, чрез командата *kill* (не може да се игнорира или променя реакцията на процеса)
10	SIGUSR1	Безусловно завършване на процеса	Сигнал, използван от потребителските процеси като средство за междупроцесна комуникация
11	SIGSEGV	Завършване на процеса със създаване на core файл	Обръщение към недопустим адрес или към адрес, за който процесът няма права
12	SIGUSR2	Безусловно завършване на процеса	Сигнал, използван от потребителските процеси като средство за междупроцесна комуникация
13	SIGPIPE	Безусловно завършване на процеса	Опит на процес да пише в програмен канал, който вече не е отворен за четене.
14	SIGALRM	Безусловно завършване на процеса	Изтичане на времето, заредено от процеса, чрез системния примитив alarm
15	SIGTERM	Безусловно завършване на процеса	Предупреждение за завършване на процеса

17	SIGCHLD	Сигналът се игнорира	Процес-дете е преминал в състояние «Завършване» или «Очакване»
20	SIGTSTP	Процеса преминава в състояние «Очакване»	Хардуерно прекъсване от калвиатурата – натискане на клавишите <Ctrl>+<z>

2. Изпращане и обработка на сигнали

В контекста на процеса се съдържа специално поле, което показва, че към процеса е изпратен сигнал. Това полето е масив битове, по един бит за всеки от сигналите в UNIX, общо 31 на брой. Когато ядрото изпрати сигнал към даден процес съответния бит, отговарящ за този вид сигнал става 1-ца. С това работата по изпращане е завършена.

Процесът може да установи, че му е изпратен сигнал (да получи сигнала) само когато се намира в състояние «Изпълнение в потребителски режим» или когато ядрото го предизвика да реагира по някакъв начин на изпратения сигнал. При получаване на сигнала съществуват три начина за реакция:

1) Процесът игнорира получения сигнал (не се предприема нищо).

2) Процесът преминава с състояние «Завършен» (това е реакцията по подразбиране за повечето от сигналите).

3) Процесът обработва сигнала по начин определен от потребителя (програмистта), като изпълнява определена потребителска функция, след което продължава изпълнението си от мястото където е бил прекъснат.

Ако процес не обработи даден сигнал, то всеки следващ сигнал от същия вид, изпратен към процеса бива загубен. Това е така, тъй като не съществува структура в контекста на процеса, която да буферира постъпващите сигнали, т.е. в даден момент процеса може да приеме и обработи най-много един сигнал от определен вид.

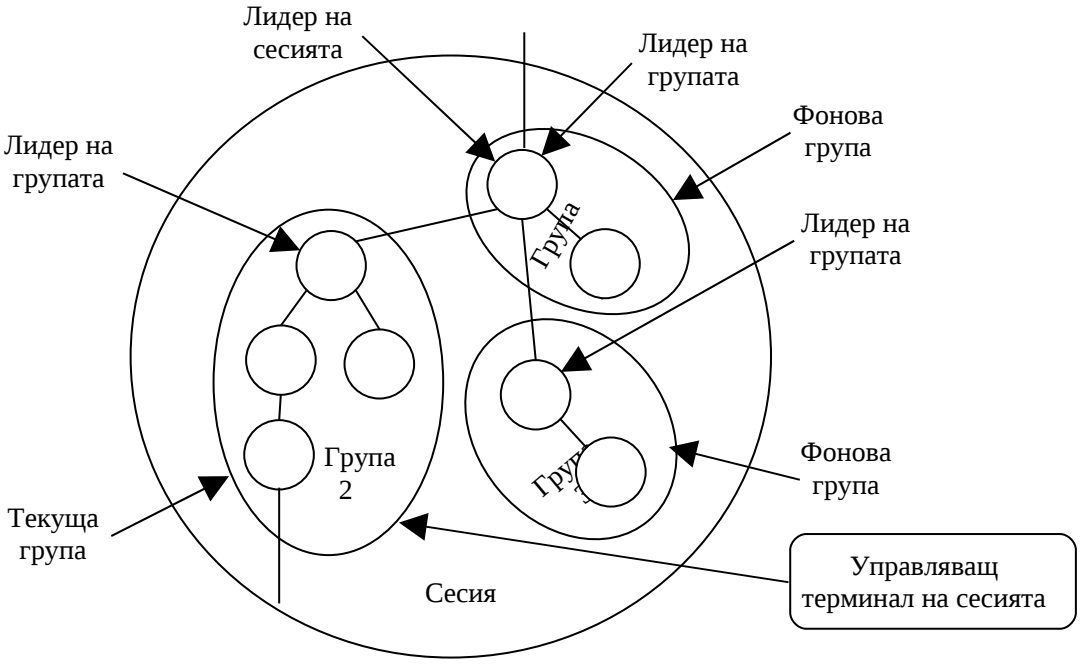
Процесите могат да блокират постъпващите сигнали от определен вид. Блокираният вид сигнали могат да бъдат изпращани към процеса, но ще бъдат получени, след тяхното деблокиране. Блокирането и деблокирането на сигнали се извършва като битовете в специален масив от битове, който е част от контекста на процеса се установяват в 1 или 0.

2. Групи от процесите, сесия, лидер на група, лидер на сесия

Както вече бе споменато всички процеси в системата са свързани и образуват дърво или гора от дървета. В качеството на възли на дървото са

процесите, а връзките между възлите показват отношението родител-дете между тях. Всяко дърво е прието да се разделя на групи от процеси или семейства (фиг. 3.1).

Групата от процеси включва един или повече процеса и съществува, докато в нея присъства поне един процес. Всеки процес задължително бива включен в някоя група. При създаването на нов процес, той попада в тази група от процеси, в която се намира и неговия родител. Всеки процес може да мигрира самостоятелно в друга група или да бъде преместен от друг процес (в зависимост от версията на UNIX). Съществуват множество системни извиквания, които могат да бъдат приложени не към един конкретен процес, а към всички процеси в дадена група. Ето защо обединяването на процесите в групи, зависи от това как се предполага, че ще бъдат използвани.



Фиг. 3.1. Групиране на процесите в UNIX

От своя страна, групите от процеси се обединяват в сесии (семейни кланове – от гледна тока на родствените връзки). Първоначално понятието сесия е въведено в UNIX за логически обединени групи от процеси, създадени в резултат от всеки вход и последваща работа на потребител в системата. Ето защо, с всяка сесия в системата може да бъде свързан терминал, наричан управляващ терминал на сесията. Обикновено, чрез

този терминал процесите от сесията могат да взаимодействат с потребителя. Сесията не може да има повече от един управляващ терминал, и обратно един терминал не може да бъде управляващ за няколко сесии. В същото време могат да съществуват сесии, които нямат управляващ терминал.

Всяка група от процеси в системата получава собствен уникален номер. Този номер може да бъде извлечен с помоща на системното извикване *getpgid()*, което връща като резултат не само номера на групата от процеси за извикващия го процес, но и за процесите от неговата сесия. Това системно извикване се различава при BSD базираните UNIX системи и тези които са System V базирани. Вместо извикването *getpgid()* в някой системи съществува системното извикване *getpgrp()*, което връща единствено номера на групата на текущия процес.

ИМЕ

getpgid

ПРОТОТИП

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpgid(pid_t pid);
```

ОПИСАНИЕ

Системното извикване връща като резултат идентификатора на групата от поцеси, а която принадлежи процеса с идентификатор pid.
Стойността на pid се отнася единствено за текущия процес или за процеси от неговата сесия. При други стойности на pid системното извикване връща стойност -1.
Типът данни pid_t е синоним на един от целочислените типове в езика C.

ИМЕ

getpgrp

ПРОТОТИП

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpgrp(void);
```

ОПИСАНИЕ

Системното извикване getpgrp връща идентификатора на групата от процеси, на която принадлежи текущия процес.
Типът данни pid_t е синоним на един от целочислените типове в езика C.

Преминаването на един процес в друга група е възможно, като едновременно с неговото създаване се изпълни системното извикване *setpgid()*. Процесът може да премести или сам себе си в друга група, или свой процес-дете, който не е изпълнил системно извикване *exec()*, т.е. не

е извикал за изпълнение друга програма. При определени стойности на параметрите на системното извикване се създава нова група от процеси с идентификатор, съвпадащ с идентификатора на преместваният процес. Първоначално тази група съдържа единствено преместения процес. Това е единственият способ за създаване на нова група. Това показва, защо идентификаторите на групи в ситемата са уникални. Преход на процес в друга група без създаването на нова е възможно единствено в рамките на една сесия.

В някой разновидности на UNIX системното извикване *setpgid()* липсва, а вместо него се използва системното извикване *setpgrp()*, посредством което може да се създава нова група от процеси с идентификатор, съвпадащ с идентификатора на текущия процес и преместване на текущия процес в нея.

ИМЕ

```
setpgid()
```

ПРОТОТИП

```
#include <sys/types.h>
#include <unistd.h>
int setpgid(pid_t pid, pid_t pgid);
```

ОПИСАНИЕ

Системното извикване setpgid служи за прехвърляне на процес от една група в друга, а също и за създаване на нова група от процеси.
Параметърът pid представлява идентификатор на преместваният процес, а параметърът pgid – идентификатор на приемната група от процеси.
Не всички комбинации от параметри са разрешени. Процесът може или сам да се премести в друга група или да премести свой процес-дете, който не е изпълнил системното извикване exec(), т.е. не стартирал за изпълнение другуа програма.
Ако параметърът pid е равен на 0, то се счита, че процесът премества сам себеси.
Ако параметърът pgid е равен на 0, то в Linux се счита, че процесът се премества в група с идентификатор, съвпадащ с идентификатора на процеса, определен от първия параметър.
Ако стойностите на параметрите pid и pgid са равни, то се създава нова група с идентификатор, съвпадащ с идентификатора на преместваният процес. Преход в друга група без създаване на нова е възможен единствено в рамките на една сесия.
В нова група не може да се премества процес, който е лидер на група, т.е. процес, чийто идентификатор съвпада с идентификатора на собствената му група.
Типът данни pid_t се явява синоним за един от целочислените типове в езика C.

ВРЪЩАНИ СТОЙНОСТИ

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

ИМЕ

```
setpgrp()
```

ПРОТОТИП

```
#include <sys/types.h>
#include <unistd.h>
int setpgrp(void);
```

ОПИСАНИЕ

Системното извикване setpgrp служи за преместване на текущия процес в новосъздадена група от процеси, чийто идентификатор съвпада с идентификатора на текущия процес.

ВРЪЩАНИ СТОЙНОСТИ

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

Процесът, чийто идентификатор съвпада с идентификатора на неговата група се нарича лидер на групата. Едно от ограниченията на разгледаните системни извиквания *setpgid()* и *setpgrp()* се състои в това, че лидерът на група не може да се премества в друга група.

Всяка сесия в системата също има собствен номер. За неговото извличане може да се използва системното извикване *getsid()*. В различните версии на UNIX за него съществуват различни ограничения. В Linux такива ограничения отсъстват.

ИМЕ

getsid()

ПРОТОТИП НА СИСТЕМНОТО ИЗВИКВАНЕ

```
#include <sys/types.h>
#include <unistd.h>
pid_t getsid(pid_t pid);
```

ОПИСАНИЕ НА СИСТЕМНОТО ИЗИВКВАНЕ

Системното извикване връща идентификатора на сесията за процеса с идентификатор pid. Ако параметъра pid в равен на 0, то се връща идентификатора на сесиата за текущия процес.
Типът данни pid_t се явява синоним за един от целочислените типове в езика C.

Използването на системното извикване *setsid()* води до създаване на нова група, състояща се единствено от процеса, който го е изпълнил (той става лидер на новата група), и нова сесия, чийто идентификатор съвпада с идентификатора на процеса, изпълнил извикването. Такъв процес се нарича лидер на сесията. Това системно извикване може да се изпълни единствено от процес, който не е лидер на група.

ИМЕ

setsid()

ПРОТОТИП НА СИСТЕМНОТО ИЗВИКВАНЕ

```
#include <sys/types.h>
#include <unistd.h>
int setsid(void);
```

ОПИСАНИЕ НА СИСТЕМНОТО ИЗИВКВАНЕ

Това системно извикване може да се изпълни единствено от процес, който не е лидер на група, т.е. от процес, чийто идентификатор не съвпада с идентификатора на неговата група. Използването на системното извикване setsid() води до създаване на нова група, състояща се единствено от процеса, който го е изпълнил (той става лидер на новата група), и нова сесия, чийто идентификатор съвпада с идентификатора на процеса, изпълнил извикването.

ВРЪЩАНИ СТОЙНОСТИ

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

Ако една сесия има управляващ терминал, то този терминал задължително се причислява към някоя група от процеси, участваща в сесията. Такава група от процеси се нарича текуща група за дадената сесия. Всички процеси, намиращи се в текущата група, могат да извършват операциите за вход-изход, използвайки управляващия терминал. Всички останали групи в сесията се наричат фонови групи, а процесите намиращи се в тях – фонови процеси. При опит за вход-изход от страна на фонов процес, посредством управляващия терминал, този процес получава сигнал, който води до прекратяване на неговата работа. Предаването на управляващия терминал от една група на друга може да извърши единствено лидера на сесията. Трябва да се отбележи, че в сесия без управляващ терминал всички процеси са фонови.

При завършване на изпълнението на процеса – лидер на сесията, всички процеси от текущата група получават сигнал SIGHUP, който при стандартна обработка води до тяхното завършване. По този начин след прекратяване работата на лидера на сесията, в общия случай, работа продължават единствено фоновите процеси.

Процесите, част от текущата група за дадена сесия, могат да получават сигнали, генерирани от терминала при натискането на определени клавиши – *SIGINT* при натискането на клавиши *<ctrl> + <c>*, и *SIGQUIT* при клавиши *<ctrl> + <4>*. Стандартната реакция на тези сигнали е завършване работата на процеса (с образуване на core файл за сигнала *SIGQUIT*).

Необходимо е да се поясни понятието “ефективен идентификатор на потребителя”. Всеки потребител в системата има свой собствен идентификатор – *UID*. Всеки процес стартиран от потребителя използва този *UID*, за да определи своите пълномощия. Съществува случай, когато в изпълнимия файл са установени определени атрибути, при което процесът

вижда себе си като процес стартиран от друг потребител. Идентификатора на потребителя, чиито пълномощия използва процесът се нарича ефективен идентификатор на потребителя – *EUID*. С изключение на описания случай, ефективния идентификатор на потребителя съвпада с идентификатора на потребителя създал процеса.

3. Системно извикване kill() и команда kill

Комадната *kill* и системното извикване *kill()* са двата източника на сигнали в ОС достъпни за потребителя. Командата *kill* се използва в следната форма:

```
kill [-номер] pid
```

Параметърът *pid* е идентификатора на процеса, за който е предназначен сигнала, а *-номер* е номера на сигнала, който се изпраща към указания с *pid* процес. Изпращането на сигнали (ако потребителя няма администраторски привилегии) е възможно само към процеси, за които ефективния идентификатор на потребителя съвпада с идентификатора на потребителя изпращащ сигнала. Ако параметърът *–номер* се пропусне, то се изпраща сигнал *SIGTERM*, чийто номер е *15*. Реакцията по подразбиране на процесът получил този сигнал е да завърши своята работа.

ИМЕ

kill

СИНТАКСИС

```
kill [-signal] [--] pid
kill -l
```

ОПИСАНИЕ

Командата kill е предназначена за предаване на сигнали към един или няколко процеса в рамките на потребителските пълномощия.
Параметърът pid определя процеса или процесите, които следва да получат сигнала. Той може да бъде зададен посредством един от следните четири способа:
1. Посредством положително число n > 0 – определя идентификатора на процеса, който следва да получи сигнала.
2. Посредством число равно на 0 – сигналът ще бъде изпратен до всички процеси в текущата група за дадения управляващ терминал.
3. Посредством число равно на -1, предшествано от “--“. Така сигналът ще бъде изпратен (ако правомощията на потребителя разрешават) до всички процеси, които имат идентификатор > 1.
4. Посредством отрицателно число n < 0, където n не е равно на -1, предшествано от “--“. Така сигналът ще бъде изпратен до всички процеси, в групата с идентификатор |-n|.
Параметърът -signal определя типа на изпращания сигнал. Той може да се задава в числова или символна форма, например -9 или -SIGKILL. Ако този параметър се изпусне, се изпраща сигнал по подразбиране SIGTERM. Сигнали могат да се изпращат (ако потребителят е без права на администратор) само към процеси за

които ефективния идентификатор на потребителя съвпада с идентификатора на потребителя, изпращащ сигнала.
Опцията -l се използва за получаване на списъка на наличните сигнали, в символна и числова форма.
В много операционни системи съществуват и допълнителни опции за командата kill.

При използване на системното извикване *kill()* изпращането на сигнал (ако потребителят няма права на администратор) е възможно само към процеси, за които ефективния идентификатор на потребителя съвпада с ефективния идентификатор на потребителя за процеса, изпращащ сигнала.

ИМЕ

kill()

ПРОТОТИП

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int signal);
```

ОПИСАНИЕ

Системното извикване kill() е предназначено за предаване на сигнали към един или няколко процеса в рамките на потребителските пълномощия.
Изпращането на сигнали (ако потребителя няма администраторски привилегии) е възможно само към процеси за които ефективния идентификатор на потребителя съвпада с ефективния идентификатор на потребителя за процеса изпращащ, сигнала.
Аргументът pid описва за кого е предназначен сигнала, а аргументът sig – какъв сигнал се изпраща:
Ако pid > 0 и sig > 0, сигналът с номер sig (ако привилегиите позволяват) се изпраща на процес с идентификатор pid.
Ако pid = 0, а sig > 0, сигналът с номер sig се изпраща до всички процеси в групата, на която принадлежи и изпращащият процес.
Ако pid = -1, sig > 0 и изпращащият процес не е процес принадлежащ на суперпотребителя, сигналът се изпраща до всички процеси в системата, за които идентификаторът на потребителя съвпада с ефективния идентификатор на потребителя за процеса, изпращащ сигнала.
Ако pid = -1, sig > 0 и изпращащият процес е процес, принадлежащ на суперпотребителя, сигналът се изпраща до всички процеси в системата с изключение на системните процеси (т.е. всички освен процесите с pid = 0 и pid = 1).
Ако pid < 0, но не е –1 и sig > 0, то сигналът се изпраща до всички процеси в групата, чийто идентификатор е равен по абсолютна стойност на аргумента pid (ако привилегиите позволяват).
Ако sig = 0, то се извършва проверка за грешка и не се изпраща сигнал, тъй като всички сигнали имат номера > 0.

ВРЪЩАНИ СТОЙНОСТИ

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

4. Особености при изпращане на терминални сигнали до процеси от текущата и фоновите групи

В програмата от листинг 3.1, един процес поражда процес-дете и двата процеса влизат в безкраен цикъл.

```
/* Програма за илюстрация на понятията
група от процеси, сесия, фонова група и т.н. */
#include <unistd.h>
```

```
int main(void){
    (void)fork();
    while(1);
    return 0;
}
```

Листинг 3.1. Програма (3_1.c) за илюстрация на понятията група от процеси, сесия, фонова група и т.н.

Командата *ps* с опции *–e* и *j*, позволява да се получи информация за всички процеси в системата, включително идентификатор на процеса, идентификатор на групата и сесията, управляващия терминал и групата на управляващия терминал. Командата "`ps -e j`" (обърнете внимание на наличието на интервал между буквите `e` и `j`!!!) извежда списък на всички процеси в системата. Колоната *PID* съдържа идентификаторите на процесите, колоната *PGID* – идентификатора на групата на която принадлежат, колоната *SID* – идентификатора на сесията, колоната *TTY* – номерът съответстващ на управляващия терминал, колоната *TPGID* (може да липсва в някой от версиите на UNIX) – на коя група от процеси принадлежи управляващия терминал.

Задача 1: Въведете програмата от листинг 3.1, компилирайте и я изпълнете. Стартирайте командата *"ps –e j"* в друг терминален прозорец. Анализирайте изведените стойности на идентификаторите на групите от процеси, сесиите, мястото на управляващия терминал, текущата и фоновите групи. Убедете се, че процеса на стартираната програма се отнася към текущата група в сесията. Проверете реакцията на текущата група на сигнала *SIGINT* – клавишна комбинация <*CTRL*> + <*C*> – и *SIGQUIT* – клавишна комбинация <*CTRL*> + <*4*>.

Задача 2: Стартирайте програмата от листинг 3.1 във фонов режим, посредством командата *"a.out &"*. Анализирайте изведените стойности на идентификаторите на групите от процеси, сесиите, мястото на управляващия терминал, текущата и фоновите групи. Убедете се, че процеса на стартираната програма се отнася към фонова група в сесията. Проверете реакцията на фоновта група на сигнала *SIGINT* – клавишна комбинация <*CTRL*> + <*C*> – и *SIGQUIT* – клавишна комбинация <*CTRL*> + <*4*>. Унищожете процеса на стартираната програма с помоща на командата *kill*.

Задача 3: Използвайте отново програмата от листинг 3.1 и я стартирайте за изпълнене в текущата група. Отстранете лидера на сесията за стартираната

програма. Анализирайте изведените стойности на идентификаторите на групите от процеси, сесиите, мястото на управляващия терминал, текущата и фоновите групи, унищожете лидера на сесията за процеса на стартираната програма. Убедете се, че всички процеси в текущата група за сесията са прекратили своята работа.

Задача 4: Стартирайте програмата от листинг 3.1 във фонов режим. Отново отстранете лидера на сесията за стартираната програма. Убедете се, че фоновата група продължава да работи. Унищожете процеса на стртираната програма.

5. Системно извикване signal(). Потребителска функция за обработка на получен сигнал

Един от способите за изменение на поведението на процес при получаване на сигнал в операционната система UNIX е използването на системното извикавне *signal()*.

ИМЕ

```
signal()
```

ПРОТОТИП

```
#include <signal.h>
void (*signal (int sig,
void (*handler) (int)))(int);
```

ОПИСАНИЕ

Системното извикване signal служи за изменение на подразбиращата се реакцията на процес при получаване на какъвто и да било сигнал.
Функцията signal, връща указател на функция с един параметър от тип int, която от своя страна не връща нищо и има два параметъра: параметър sig от тип int и параметър handler, служещ за указател на нищо не връщащата функция с едни параметър от тип int.
Параметърът sig е номера на сигнала, чиято обработка предстои да се измени.
Параметърът handler описва новия способ за обработката на сигнала –може да бъде указател към потребителска функция за обработка на сигнала, специалната стойност SIG_DFL или специалната стойност SIG_IGN. Специалната стойност SIG_IGN позволява процеса да игнорира постъпващия сигнал с номер sig, а специалната стойност SIG_DFL – за възстановяване на подразбиращата се реакция на процеса за дадения сигнал.

ВРЪЩАНА СТОЙНОСТ

Системното извикване връща указател към стария способ за обработка на сигнала, стойността на който може да се използва за възстановяване на стария способ за обработка в случаи на необходимост.

Това системно извикване има два параметра: единият задава номера на сигнала, за който процесът следва да измени реакцията си, а втория определя, как ще се осъществи нейното изменение. При първия вариант, процеса реагира на сигнал (вижте т. 1 "Сигнали. Възникване и обработка на сигналите ") като го игнорира. В този случай втория параметър приема

специалната стойност *SIG_IGN*. Например, ако трябва да се игнорира сигналът *SIGINT*, след определено място в работата на програмата, на това място в програмата трябва да се използва конструкцията:

```
(void) signal(SIGINT, SIG_IGN);
```

При втория вид реакция, процеса възстановява обработката по подразбиране за тази цел стойността на параметъра трябва да бъде *SIG_DFL*. При третия вариант стойността на параметъра представлява указател към потребителска функция, която да обработи сигнала. Вида на нейния прототип трябва да бъде:

```
void *handler(int);
```

Примерна конструкция при потребителска обработка на сигнала *SIGHUP.*

```
void *my_handler(int nsig) {
    <обработка на сигнала>
}
int main() {
    ...
    (void)signal(SIGHUP, my_handler);
    ...
}
```

Номера на възникналия сигнал се предава в потребителската функция като праметър (в този случай – параметърът *nsig*), така че една и съща функция може да се използва за обработка на няколко сигнала.

6. Програма, игнорираща сигнала SIGINT

```
/* Програма, игнорираща сигнала SIGINT */
#include <signal.h>
int main(void){
    /* Задава се процеса да игнорира сигнала SIGINT*/
    (void)signal(SIGINT, SIG_IGN);
    /*От това място насетне процесът ще игнорира
      възникването на сигнал SIGINT */
    while(1);
    return 0;
}
```

Листинг 3.2. Програма (3_2.c), игнорираща сигнала SIGINT

Тази програма не извършва нищо полезно, освен да преустанови реакцията на калвишната комбинация *<CTRL> + <C>*, игнорирайки възникналия сигнал, оставайки по този начин в своя безкраен цикъл.

Задача 5: Въведете, компилирайте и изпълнете тази програма, убедете се, че при клавишната комбинация *<CTRL> + <C>*, тя не реагира, а реакция при натискане на *<CTRL> + <4>* не се е променила.

Задача 6: Модифицирайте програмата от листинг 3.2, така че да престане да

реагира и на натискането на <*CTRL*> + <*4*>. Компилирайте и изпълнете програмата. Убедете се, че липсва реакция на посочените клавишни комбинации. Преустановте работата на програмата в друг терминал с командата *kill*.

7. Програма с потребителска обработка на сигнала SIGINT

```
/* Програма с потребителска обработка на сигнала SIGINT */
#include <signal.h>
#include <stdio.h>
/* Функцията my_handler за потребителска обработка на сигнала */
void my_handler(int nsig){
    printf("Receive signal %d,
           CTRL-C pressed\n", nsig);
}
int main(void){
    /* Задаване на вида на реакцията при сигнал SIGINT */
    (void)signal(SIGINT, my_handler);
    /* От това място насетне, процесът отпечатва съобщение
       за възникване на сигнал SIGINT */
    while(1);
    return 0;
}
```

Листинг 3.3. Програма (3_3.c) с потребителска обработка на сигнала SIGINT

Тази програма се отличава от програмата в листинг 3.2 по това, че в нея е въведена обработка на сигнала *SIGINT* с помоща на потребителска функция.

Задача 7: Въведете, компилирайте и изпълнете програмата, проверете найната реакция при натискане на <*CTRL*> + <*C*> и при <*CTRL*> + <*4*>.

Задача 8: Модифицирайте програмата от листинг 3.3 така, че да отпечатва съобщение при клавишната комбинация <*CTRL*> + <*4*>. Използвайте една и съща функция за обработката на сигналите *SIGINT* и *SIGQUIT*. Компилирайте и изпълнете програмата, проверете нейната работа. Премахнете програмата с помоща на друг терминал и командата *kill*.

8. Възстановяване на подразбиращата се реакция на сигнал

В горните примери не се взема предвид връщаната стойност от системното извикване *signal()*. При своето изпълнение това системно извикване връща указател към предходната функция, обработвала сигнала, което позволява да се възстанови предходно зададената реакция на сигнал. Програмата *3_4.c*, връща първоначалната реакция на сигнала *SIGINT* след 5 потребителски обработки.

```
/* Програма с потребителска обработка на сигнала SIGINT с връщане
  към първоначалната реакция за този сигнал след 5 негови обработки */
#include <signal.h>
```

```
#include <stdio.h>
int i=0; /* Брояч на направените обработки */
void (*p)(int); /* Указател, в който се записва адреса на
                   предходната функция за обработка */
/* Функцията my_handler за потребителска обработка на сигнала */
void my_handler(int nsig){
    printf("Receive signal %d, CTRL-C pressed\n", nsig);
    i = i+1;
    /* След 5-тата обработка се възстановява
        първоначалната реакция към сигнла */
    if(i == 5) (void)signal(SIGINT, p);
}
int main(void){
    /* За да възстанови своята реакцията към сигнала SIGINT,
процесът запомня адреса на предходната функция за обработка */
    p = signal(SIGINT, my_handler);
    /* От това място насетне процесът 5 пъти ще
отпечата съобщение за възникването на сигнал SIGINT */
    while(1);
    return 0;
}
```

Листинг 3.4. Програма (3_4.c) с потребителска обработка на сигнала SIGINT

Задача 9: Въведете, компилирайте и изпълнете програмата.

9. Сигнали SIGUSR1 и SIGUSR2

В операционата система UNIX съществуват два сигнала, източниците на които могат да бъдат единствено системното извикване *kill()* или командата *kill* – това са сигналите *SIGUSR1* и *SIGUSR2.* Обикновенно те се използват за предаване на информация за предстоящи събития от един потребителски процес към друг в качеството им на сигнално средство за връзка.

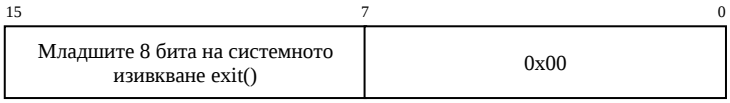
10. Завършване на породен процес

В т. 7 от упражнение 2 бе изяснено, че ако процес-дете завърши своята работа преди процеса-родител, и процеса-родител не е указал явно, че не е заинтересуван от получаването на информация за статуса на завършилия процес-дете, то завършващия процес не изчезва окончателно от системата. той остава в състояние завършил (зомби-процесс) или до завършване на процеса-родител, или докато, родителя благоволи да получи тази информация.

За да получи такава информация процеса-родител може да се възползва от системното извикване *waitpid()* или от неговата опростена форма *wait()*. Системното извикване *waitpid()* позволява на процеса-родител синхронизирано да получава данни за състоянието на завършващия процес-дете. Процеса-родител или блокира до завършването

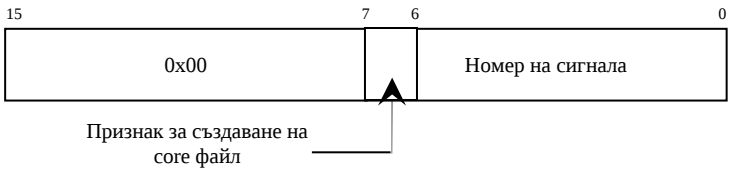
на процеса-дете, или без да блокира периодично се изивква с опцията *WNOHANG*. Данните за състоянието на завършващ процес заемат 16 бита и могат да бъдат описани по следния начин:

Ако процеса завърши с помоща на явно или неявно извикавне на функцията exit(), то данните имат следния вид:



фиг. 3.2. Вид на данните за състоянието на завършил процес след използване на функцията exit()

Ако процеса завърши след подаването на сигнал, данните за състоянието имат следния вид:



фиг. 3.3. Вид на данните за състоянието на завършил процес след изпращане на сигнал

Всеки процес-дете при завършване на своята работа изпраща до своя процес-родител специален сигнал *SIGCHLD*, за който реакцията по подразбиране е "игнориране на сигнала". Наличието на такъв сигнал съвместно със системното извикване *waitpid()* позволява да се организира асинхронно събиране на информаця за състоянието на завършващите процеси от процесите-родители.

ИМЕ

```
wait waitpid
```

ПРОТОТИП

```
#include <sys/types.h>
#include <wait.h>
pid_t waitpid(pid_t pid, int *status, int options);
pid_t wait(int *status);
```

ОПИСАНИЕ

```
Системното извикване waitpid() блокира изпълнението на текущия процес до
момента, когато: завърши породения от него процес, определен със стойността
на параметъра pid; текущия процес получи сигнал, за който е установена
реакция по подразбиране "завършване на процес"; реакцията е обработка на
сигнала от потребителска функция. Ако породения процес, зададен с параметъра
pid, към момента на системното извикване се намира в състояние завършил, то
системното извикване незабавно връща изпълнението на текущия процес без да го
блокира.
```

Параметърът pid определя породения процес, чието завършване е очаквано от процеса-родител:
- Ако pid > 0 се очаква завършването на процеса с идентификатор pid.
- Ако pid = 0 се очаква завършването на всички породени процеси в групата на която принадлежи процеса-родител.
- Ако pid = -1 се очаква завършването на всички породени процеси.
- Ако pid < 0, но не е –1 се очаква завършването на всички породени процеси в групата, чийто идентификатор е равен по абсолютна стойност на параметра pid.

Параметърът options може да приема две стойности: 0 и WNOHANG. Стойността WNOHANG указва незабавно връщане от извикването без блокиране на текущия процес.

Ако системното извикване намери завършващ породен процес с номер съответстващ на параметъра pid, то този процес се изтрива от системата, а в адресът, указан, чрез параметъра status се съхранява информация за състоянието на завършване на процеса. Параметърът status може да бъде равен на NULL, ако в определени случаи състоянието на завършване няма значение.

При намиране на завършващ процес системното извикване връща неговият идентификатор. Ако извикването е направено с опцията WNOHANG, и съществува породен процес с идентификатор равен на параметъра pid, който още не е завършил, системното извикване връща стойност 0. Във всички останали случаи системното извикване връща отрицателна стойност. Връщането от извикването, свързано с възникване на обработен от потребителя сигнал може да бъде установено по стойността на системната променлива errno == EINTR и извикването може да бъде направено отново.

Системното извикване wait се явява синоним за системното изивкване waitpid със стойности на параметрите pid = -1 и options = 0.

Използвайки системното извикване *signal()* е възможно явно да се установи игнорирането на сигнала *SIG_CHLD*, с единствента цел ситемата да се информира, че за процеса-родител е без значение как и защо завършва породения процес. В този случай отпада възникването на зомби-процеси и необходимостта от използване на системните извиквания *wait()* и *waitpid()*.

Програма *3_5.c* демонстрира асинхроното получаване на информация за състоянието на завършване на породен процес и обработката на сигнала SIGCHLD.

```
/* Програма с асинхронно получаване на информация за
състоянието на два завършващи породени процеса */
#include <sys/types.h>
#include <unistd.h>
#include <waith>
#include <signal.h>
#include <stdio.h>
/* Функция my_handler – обработваща сигнала SIGCHLD */
void my_handler(int nsig){
    int status;
    pid_t pid;
    /* Разглежда се състоянието на завършващия процес и
       едновременно се извлича неговия идентификатор */
    if((pid = waitpid(-1, &status, 0)) < 0){
        /* Ако възникне грешка се съобщава за нея и
           работата се продължава */
        printf("Some error on waitpid errno = %d\n",
            errno);
    } else {
        /* В противен случай се анализира състоянието на
           завършилия процес */
        if ((status & 0xff) == 0) {
            /* Процесът е завършил с явно или неявно
               изивикване на функцията exit() */
```

```
                printf("Process %d was exited with status %d\n",
                pid, status >> 8);
            } else if ((status & 0xff00) == 0){
                /* Процесът е завършил с помоща на сигнал */
                printf("Process %d killed by signal %d %s\n",
                pid, status &0x7f,(status & 0x80) ?
                "with core file" : "without core file");
            }
        }
    }
    int main(void){
        pid_t pid;
        /* Задава се функцията за обработка на сигнала SIGCHLD */
        (void) signal(SIGCHLD, my_handler);
        /* Поражда се Child 1 */
        if((pid = fork()) < 0){
            printf("Can\'t fork child 1\n");
            exit(1);
        } else if (pid == 0){
            /* Child 1 – завършва с код 200 */
            exit(200);
        }
        /* Процеса-родител продължава – поражда Child 2 */
        if((pid = fork()) < 0){
            printf("Can\'t fork child 2\n");
            exit(1);
        } else if (pid == 0){
                    /* Child 2 – зацикля, необходимо е да бъде изтрит
               с помоща на сигнал! */
            while(1);
        }
        /* Процеса-родител продължава – влиза в цикъл */
        while(1);
        return 0;
    }
```

Листинг 3.5. Програма 3_5.c с асинхроно получаване на информация за състоянието на два завършващи породени процеса

В тази програма родителя поражда два процеса. Единият от тях завършва с код 200, а вторият зацикля. Преди пораждането на процесите родителя задава функцията за обработка на сигнала SIGCHLD, а след пораждането на процесите-деца влиза в безкраен цикъл. В обработващата функция се извиква *waitpid()* за всеки от породените процеси. Тъй като в обработващата функция се попада, когато, който и да е от процесите завърши, системното изивкване не блокира и може да се получи информация за идентификатора на завършващия процес и причините за неговото завършване.

Задача 10: Компилирайте програмата и я изпълнете. Завършете изпълнението на втория породен процес с помоща на командата *kill* с какъвто и да било номер на сигнал. Родителският процес също е необходимо да бъде завершен с командата *kill*.

11. Надежност на сигналите

Основният недостатък на системното извикване *signal()* се явява неговата ниска надежност. В много от вариантите на операционната система UNIX установената с негова помощ потребителска функция за обработка на сигнал се изпълнява само веднъж, след което автоматично се възстановява подразбиращата се реакция към сигнала.

В системните извиквания и потребителските програми могат да съществуват критични участъци, в които е недопустимо работата на процесите да се спира от обработката на сигнали. В такива участъци може да се установи реакцията "игнориране на сигнал" с последващо възстановяване на предходната реакция, но ако сигналът все така възниква в критичния участък, то информацията за неговото възникване ще бъде безвъзвратно загубена. Това е свързано с невъзможността на процесите да приемат голям брой сигнали от един и същи тип, когато се намират в състояние готовност. Процесите немогат да определят броя на предадените към тях сигнали, а само от какъв тип са те. Този недостатък, лесно може да се илюстрира, като се промени програмата с асинхроното получаване на информация за състоянието на завършилия процес (програма *3_5.c*). В новата програма (програма *3_6.c*) процесът-родител поражда в цикъл пет нови процеса, всеки от които веднага завършва със свой собствен код, след което програмата влиза в безкраен цикъл.

```
/* Програма, илюстрираща (не)надежноста на сигналите */
#include <sys/types.h>
#include <unistd.h>
#include <waith>
#include <signal.h>
#include <stdio.h>
/* Функция my_handler – обработваща сигнала SIGCHLD */
void my_handler(int nsig){
    int status;
    pid_t pid;
    /* Разглежда се състоянието на завършващия процес и
       едновременно се извлича неговия идентификатор */
    if((pid = waitpid(-1, &status, 0)) < 0){
        /* Ако възникне грешка се съобщава за нея и
           работата се продължава */
        printf("Some error on waitpid errno = %d\n", errno);
    } else {
        /* В противен случай се анализира състоянието на
           завършилия процес */
        if ((status & 0xff) == 0) {
            /* Процесът е завършил с явно или неявно
               изивикване на функцията exit() */
            printf("Process %d was exited with status %d\n",
            pid, status >> 8);
        } else if ((status & 0xff00) == 0){
            /* Процесът е завършил с помоща на сигнал */
            printf("Process %d killed by signal %d %s\n",
```

```
                pid, status &0x7f,(status & 0x80) ?
                "with core file" : "without core file");
            }
        }
    }
    int main(void){
        pid_t pid;
        int i;
        /* Задава се функцията за обработка на сигнала SIGCHLD */
        (void) signal(SIGCHLD, my_handler);
        /* В цикъл се пораждат 5 процеса-деца */
        for (i=0; i
            if((pid = fork()) < 0){
                printf("Can\'t fork child %d\n", i);
                exit(1);
            } else if (pid == 0){
                /* Child i – завършва с код 200 + i */
                exit(200 + i);
            }
            /* Процеса-родител продължава – влиза в нова итерация */
        }
        /* Процеса-родител продължава – влиза в цикъл */
        while(1);
        return 0;
    }
```

Листинг 3.6. Програма 3_6.c, илюстрираща (не)надежността на сигналите

Въпрос: Колко на брой са очакваните съобщенията за състоянието на завършващите процеси-деца?

Задача 11: Компилирайте програма *3_6.c* и я изпълнете. Колко на брой са получаваните съобщения?

12. POSIX функции за работа със сигнали

С въвеждането на стандарта POSIX за системните извиквания в UNIX, наборът функции и системни извиквания за работа със сигнали бива съществено разширен и построен по такъв начин, че позволява временно да се блокира обработката на определени сигнали, като не се допуска тяхната загуба. Въпреки това, проблемът свързан с определяне на броя постъпващи сигнали от един и същи тип, остава актуален. (Трябва да се отбележи, че подобен проблем съществува на хардуерно ниво при апаратните прекъсвания. Процесорът често не може да определи, какво количество апаратни прекъсвания с един и същ номер възникват, до изпълнението на следващата команда.)

Някой POSIX функции и системни извиквания за работа със сигнали са: *sigemptyset(), sigfillset(), sigaddset(), sigdelset(), sigismember(), sigaction(), sigprocmask(), sigpending(), sigsuspend()*, подробна информация за които може да бъде намерена в

UNIX Manual.

Задача 12 (повишена сложност): Променете обработката на сигналите в програмата `3_6.c` (без да използвате POSIX-функции), така че, процесът-родител винаги да съобщава за състоянието на завършващия процес-дете.