ТЕМА 5. ВЗАИМОДЕЙСТВИЕ НА ПРОЦЕСИ ПОСРЕДСТВОМ СЪОБЩЕНИЯ

Преимущества и недостатъци на потоковия обмен на данни. Понятие за System V IPC. Пространство на имената. Адресация в System V IPC. Функция ftok(). Дескриптори в System V IPC. Съобщенията, като средство за връзка и средство за синхронизация на процеси. Опашки от съобщения в UNIX. Създаване на опашки от съобщения или достъп до вече съществуващи. Системно извикавне msgget(). Реализация на примитивите send и receive. Системни извиквания msgsnd() и msgrcv(). Изтриване на опашки от съобщения с помощта на командата ipcrm или системното извикване msgctl(). Мултиплексиране на съобщения. Модел за взаимодействие между процеси "клиент-сървър". Неравнопоставеност на клиента и сървъра.

1. Недостатъци на потоковия обмен на данни

Потоковите механизми с неименуван и именуван канал, разгледани в предишното упражнение имат проста реализация и са удобни за използване, но заедно с това имат и своите недостатъци:

•Не е ясно дали данните в канала са постъпили наведнъж или на порции, а освен това дали са записани от един процес или от няколко. За да могат да интерпретират данните, приемащиия и предаващия процес, трябва предварително да съгласуват своите действия.

•Предаването на данни от един процес към друг се извършва, чрез минимум две операции копиране: веднъж от адресното пространство на предаващия процес в системния буфер и втори път от системния буфер в адресното пространство на приемащия процес.

•Процесите, които си обменят информация, посредством канали, трябва да съществуват едновремено в системата, независимо, че четенето може да се осъществи значително по-късно след записа.

Недостатъците на потоковия обмен на данни са избегнати в механизмите за междупроцесна комуникация, наречени System V IPC (IPC е съкращение от interprocess communications). Тази група от механизми включва: предаване на съобщения, обща памет и семафори, които имат

сходен интерфейс и общи операции (например, освобождаване на системен ресурс).

2. Пространство на имената. Адресация в System V IPC

Всички средства за връзка в System V IPC, както и каналите (pipe и FIFO) са средства за връзка с непряка адресация. Както бе посочено в предходното упражнение, взаимодействието на неродствени процеси с помощта на средства за връзка с непряка адресация е възможно само, ако тези средства имат имена. От тази гледна точка отсъствието на име в pipe (неименования канал) позволява, единствено процеси с родствена връзка да получват информация за разположението на канала в системата и неговото състояние. И обратно, наличието на асоциирано име при FIFO (именования канал) – име на специализиран файл във файловата система – позволява на неродствени процеси да получат достъп до него, чрез интерфейса на файловата система.

Множеството от всички възможни имена за обекти от определен вид е прието да се нарича пространство на имента за съответния вид обекти. За FIFO, пространството на имената се състои от множеството на всички допустими имена на файлове във файловата система. Пространството на имента за всички System V IPC обекти се състои от множеството от стойности за типа данни `key_t` (ключ), състоящо се от цели положителни числа. Стойността на този ключ се задава опосредствено, чрез комбинацията от име на определен файл, наличен във файловата система и положително цяло число – например, номера на инстанцията на средството за връзка.

Първият компонент, който е основен при формиране на стойността на ключа, позволява да се избегне възможността двама програмисти, случайно да използват една и съща стойност на ключа и съответно едно и също средство за комуникация, което може да доведе до неподозирана комуникация между техните процеси.

Програмистът трябва да използва файл, до който взаимодействащите си процеси имат достъп с право за четене. Пълното име на този файл се преобразува в числова стойност. Ето защо, файлът не трябва да променя своето местоположение, докато протича взаимодействието на процесите.

Вторият компонент, участващ във формирането на ключа позволява на програмиста да свърже с едно и също име на файл, повече от една инстанция (екземпляр) на дадено средство за връзка. За стойност на този компонент може да се използва поредния номер на съответната инстанция.

За получаване (генериране) на стойността на ключа от описаните два компонента се използва функцията *ftok()*.

ИМЕ

ftok

ПРОТОТИП

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char *pathname, char proj);
```

ОПИСАНИЕ

Функцията ftok служи за преобразуване на името на съществуващ файл и положително цяло число, например, поредния номер на екземпляра (инстанцията) на средството за връзка, в ключ за System V IPC.
Параметърът pathname е указател към името на съществуващ файл, достъпен за процеса, извикващ функцията.
Параметърът proj е положително цяло число, характеризиращо екземпляра (инстанцията) на средството за връзка.
В случай на невъзможност да бъде генериран ключ, функцията връща отрицателна стойност, в противен случай, тя връща стойността на генерирания ключ. Типът данни key_t е 32-битово цяло число.

Необходимо е указания, чрез параметъра *pathname* файл да същества във файловата система и процесът да има право на достъп до него на ниво четене. Указания файл не трябва да променя своето местоположение на диска до момента, в който всички процеси, участващи във взаимодействието, не получат System V IPC ключа. Използването на файл, като един от компонентите за генериране на ключа, в никакъв случай не означава, че информацията, предавана с помощта на асоцииранота средство за връзка се разполага в този файл. Тази информация се съхранява вътре в адресното пространство на операционната система, а зададеното име на файл, единствено се използва от различните процеси да генерират идентични ключове.

3. Дескриптори в System V IPC

В предходното упражнение бе въведено понятието файлов дескриптор. Системните извиквания, осъществяващи операции над файлове, включително именувани и неименувани канали използват като параметър номера на елемента в таблицата на отворените файлове,

съответстващ на използвания входно-изходен поток (файл или канал) – файловия дескриптор. Използването на файлови дескриптори за идентифициране на входно-изходните потоци, вътре в процеса позволяват, върху тях да се прилага стандартния интерфейс за работа с файлове. В същото време обаче, това води до автоматично затваряне на каналите при завършване на процеса, което обяснява и един от посочените по-горе недостатъци на потоковия обмен на данни.

В System V IPC се използва различна концепция. Вместо в контекста на потребителските процеси, информацията за всички използвани в системата System V IPC средства се съхранява в ядрото на операционната система. При създаване на ново средство за връзка или получаване на достъп до съществуващо такова, процесът получава неотрицателно цяло число – дескриптор (идентификатор), който еднозначно идентифицира това средство за връзка в цялата изчислителна система. Този дескриптор се използва в качеството му на параметър във всички системи извиквания, осъществяващи операции над съответното System V IPC средство. Така се избягва един от най-съществените недостатъци, присъщи на потоковите средства за връзка – необходимостта взаимодействащите си процеси да съществуват едновременно в ОС. В същото време обаче, е необходимо повишено внимание по отношение на актуалността на получаваните в процеса данни. Тъй като е възможно да бъдат приети, случайно останали в механизма за комуникация стари данни.

4. Съобщенията, като средство за връзка и средство за синхронизация на процеси

Съобщенията са способ за взаимодействие на процеси, посредством линия за връзка. В нея предаваната информация има предварително дефинирана структура. Тя дава възможност на процеса, приемащ данните да определи къде свършва дадена порция информация и къде започва следващата. Така, може да се организира двупосочно предаване на данни между няколко процеса, като се използва само една линия (опашка). Освен това, модела за взаимодействие посредством съобщения притежава вграден механизъм за синхронизация между процеси. Той осигурява средства за взаимоизключване (ако процесите опитат едновременен достъп

до буфера) и блокиране (ако процес опита да чете от празен или да запише в пълен буфер) на вазимодействащите си процеси.

5. Опашки от съобщения в UNIX

Опашките от съобщения, както и останалите средства, влизащи в състава на System V IPC (семафори и поделена памет) са средства с непряка адресация. Това означава, че е необходимо да се инициализират, преди да бъдат използвани. Инициализацията включва генериране на ключ с помощта на функцията *ftok()*. Множеството от стойности на този ключ определят пространството на имената за опашките от съобщения.

За запис в опашка и четене от нея се използват примитивите *send* и *receive*. Те се реализират със съответни системни извиквания, чиито параметри включват IPC-дескриптора на опашката от съобщения, еднозначно идентифициращ я в цялата изчислителна система.

Опашките от съобщения се разполагат в адресното пространство на ядрото на операционната система във вид на еднопосочни списъци с ограничен капацитет (обемът на информацията, съхраняваща се във всяка опашка е ограничен). Всеки елемент в списъка представлява едно отделно съобщение. Съобщенията притежават атрибут, определящ типа им. Той участва при формиране на реда, в който съобщенията се четат:

• Съобщенията се получават (четат) в реда на постъпването им (организация FIFO), независимо от техния тип;

• Първо се получават (четат) съобщенията от определен тип, в реда на постъпването им (организация FIFO);

• Първо се получават (четат) съобщенията, чиито тип, не превишава определена зададена стойност, в реда на тяхното пристигане.

Реализацията на примитивите *send* и *receive* обезпечава, скрито от потребителя взаимоизключване при поместване на съобщения в опашката или при получаването им от нея. Освен това процесите се блокират при опит за четене (примитив *receive*), ако опашката е празна или отсъстват съобщения от определен тип, както и при опит за запис (примитив *send*), ако в опашкта няма свободно място.

Опашките от съобщения, както и другите средства в System V IPC, позволяват да се организира взаимодействие между процеси, за които не е

задължително да се намират едновременно в изчислителната система.

6. Създаване и достъп до съобщения

За създаване на опашка от съобщения, асоциирана с определен ключ, или достъп до вече съществуваща опашка, според зададен ключ се използва системното извикване *msgget()*, което връща стойността на IPC-дескриптора за тази опашка. Това извикване (както предстои да бъде разгледано) е аналог на системното извикване *shmget()* при използване на механизма на поделена памет и *semget()* при използване на механизма на семафорите.

ИМЕ

msgget

ПРОТОТИП

```
#include <types.h>
#include <ipc.h>
#include <msg.h>
int msgget(key_t key, int msgflg);
```

ОПИСАНИЕ

Системното извикване msgget е предназначено за получаване на достъп до опашка от съобщения, характеризираща се със своя ключ. В случай на успешно приключване, то връща нейния System V IPC дескриптор (цяло неотрицателно число, което еднозначно характеризира опашката в цялата изчислителна система), той е необходим за последващата работа с опашката.
Параметърът key е System V IPC ключа, който идентифицира опашката от съобщения. На практика, ключа е името на опашката в пространството на System V IPC имената. Стойността на параметъра key може да се получи като се използва функцията ftok(). Също така, той може да приеме специалната стойност IPC_PRIVATE. Използването на стойността IPC_PRIVATE води до опит за създаване на нова опашка с ключ, който не съвпада с нито един от ключовете на съществуващите опашки и който не може да бъде получен с помоща на функцията ftok() при нито една от комбинациите на нейните параметри.
Параметърът msgflg е флаг, който се използва само при създаване на нова опашка и определя правата на различните потребители за достъп до нея. Осен това, флага се използва за определяне на необходимостта от създаване на нова опашка и определяне на поведението на системното извикване при нейното създаване. Стойността на този параметър се задава като комбинация (с помоща на побитовата операция или – "|") от следните предварително определени стойноти и правата за достъп в осмичен вид:
IPC_CREAT – ако указаната от ключа опашка не съществува, тя трябва да бъде създадена;
IPC_EXCL – използва се заедно с флага IPC_CREAT. При съвместното им използване и наличието на масив с указания ключ, достъп до опашката не се осъществява и се констатира възникване на грешка, при това променливата errno, описана във файла <errno.h>, приема стойността EEXIST;
0400 – право за четене за потребителя, създал опашката;
0200 – право за запис за потребителя, създал опашката;
0040 – право за четене за групата на потребителя, създал опашката;
0020 – право за запис за групата на потребителя, създал опашката;
0004 – право за четене за всички останали потребители;
0002 – право за запис за всички останали потребители;
Количеството информация съхранявано в опашката е ограничено. То може да се променя от системния администратор. Текущата стойност на огранчението може да се провери с командата:
ipcs -l

ВРЪЩАНИ СТОЙНОСТИ

Системното извикване връща стойнстта на System V IPC дескриптора, при нормално завършване и стойност -1 при възникване на грешка.

7. Реализация на примитивите send и receive

За изпълнение на примитива *send* (запис в опашката) се използва системното извикване *msgsnd()*, което копира потребителско съобщение в опашка с определен IPC-дескриптор. При използване на това извикване е необходимо да се обърне особено внимание на следното:

•Типът *struct msgbuf* не е типа данни използван за потребителските съобщения. Той е само шаблон за създаване на такива типове. Потребителят е длъжен сам да създаде структура за своите съобщения. Първото поле в нея трябва да бъде променлива от тип *long*. В тази промелива се записва положителна стойност, определяща типа на съобщенията.

•Третият параметър на извикването – дължина на съобщението – указва, какъв е размера на полезната информация в съобщението, т. е. размера на данните разположени след полето за тип на съобщението. Размерът може да бъде равен на *0* в случай, че се цели съобщението да се използва единствено като сигнално средство за връзка.

•В това упражнение като правило винаги се задава стойност нула за последния параметър на извикването – параметъра *flag* – което води до блокиране на процеса при отсъствие на свободно място в опашката от съобщения.

ИМЕ

msgsnd

ПРОТОТИП

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, struct msgbuf *ptr, int length, int flag);
```

ОПИСАНИЕ

Системното извикване msgsnd е предназначено за поместване на съобщения в опашка от съобщения, т. е. то е реализация на примитива send.
Параметърът msqid е System V IPC дескриптора на опашката, в която се изпраща съобщението. Неговата стойност е тази върната от системнто извикване msgget() при създаване на опашката или при достъпа до нея по зададен ключ.
Структурата struct msgbuf е описана във файла <sys/msg.h> като

```
struct msgbuf {long mtype;
               char mtext[1];};
```

Тя представлява само шаблон за структуриране на потребителските съобщения. Първият елемент в едно потребителско съобщение според тази структура задължително трябва да е от тип long и да съдържа типа на съобщението, след

което трябва да следва информационната част, която съдържа същинското съобщение. Теоретично, тя може да има произволна дължина (на практика в Linux, тя е ограничена до размер от 4080 байта и може да бъде намалена от системния администратор). Например:

```
struct mymsgbuf {long mtype;
                 char mtext[1024];
                } mybuf;
```

При това информацията съвсем не е задължително да бъде текстова, например:

```
struct mymsgbuf {long mtype;
                 struct {int iinfo;
                         float finfo;
                        } info;
                } mybuf;
```

Типът на съобщението трябва да бъде цяло положително число. Действителната дължина на полезната част от информацията (т. е. информацията, разположена в структурата след типа на съобщението) трябва да бъде предадена на системното извикване като стойност на параметъра length. Този параметър трябва да бъде 0, ако самото наличие на съобщение е полезно като информация. Системното извикване копира съобщението, разположено на адреса указан от параметъра ptr, в опашката зададена с дескриптора msqid.

Параметърът flag може да приема две стойности: 0 и IPC_NOWAIT. Ако стойността на флага е 0 и в опашката няма място за съобщението, то системното извикване блокира процеса докато се освободи достатъчно място. Ако стойността на флага е IPC_NOWAIT, то системното извикване не се блокира, а се констатира възникване на грешка и в променливата errno, описана във файла <errno.h> се записва стойност EAGAIN.

ВРЪЩАНИ СТОЙНОСТИ

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

Примитивът *receive* се реализира от системното извикване *msgrcv()*. При използване на това системно извикване е необходимо да се обърне внимание на следното:

•Типът данни *struct msgbuf*, както и при извикването *msgsnd()* е само шаблон за потребителския тип данни.

•Според стойността на параметъра *type* – нулева, положителна и отрицателна се определя начина за получаване на съобщенията от опашката. Точната стойност на типа на избраното съобщение може да се определи от съответното поле в структурата, в която системното извикване копира съобщението.

•Системното извикване връща дължината само на полезната част от копираната информация, т. е. на информацията, разположена в структурата след полето за типа на съобщението.

•След получаване на съобщението, то се изтрива от опашката.

•Посредством параметъра *length* се указва максималната дължина на полезната част от информацията, която може да бъде разположена в структурата, адресирана от параметъра *ptr*.

•В това упражнение, като правило, се използва нулева стойност за

параметъра *flag*, което води до блокиране на процеса в случай, че в опашката липсва съобщение от заявения тип или до възникване на грешка в случай, че дължината на информативната част на избраното съобщение превишава дължината на определената от параметъра *length*.

ИМЕ

msgrcv

ПРОТОТИП НА СИСТЕМНОТО ИЗВИКВАНЕ

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv(int msqid, struct msgbuf *ptr, int length, long type, int flag);
```

ОПИСАНИЕ НА СИСТЕМНОТО ИЗВИКВАНЕ

Системното извикване msgrcv е предназначено за получаване на съобщения от опашка от съобщения, т. е. то е реализация на примитива receive.
Параметърът msqid е System V IPC дескриптора за опашката, от която трябва да бъде получено съобщението. Неговата стойнист е тази върната от системнто извикване msgget() при създаване на опашката или при достъп до нея по зададен ключ.
Параметърът type определя способа за избор на съобщения от опашката по следния начин:
Ако type=0 се избира първото постъпило в опашката съобщение, независимо от наговия тип;
Ако type=n се избира първото постъпило в опашката съобщение от тип n;
Ако type=-n се избира първото постъпило в опашката съобщение, чийто тип е по-малък или равен по абсолютна стойност от -n
Структура struct msgbuf е описана във файла <sys/msg.h> като

```
struct msgbuf {long mtype;
               char mtext[1];};
```

Тя представлява само шаблон за структуриране на потребителските съобщения. Първият елемент в едно потребителско съобщение, според тази структура задължително трябва да е от тип long и да съдържа типа на съобщението, след което трябва да следва информационната част, която съдържа същинското съобщение. Теоретично, тя може да има произволна дължина (на практика в Linux, тя е ограничена до размер от 4080 байт и може да бъде намалена от системния администратор). Например:

```
struct mymsgbuf {long mtype;
                 char mtext[1024];
                } mybuf;
```

При това информацията съвсем не е задължително да бъде текстова, например:

```
struct mymsgbuf {long mtype;
                 struct {int iinfo;
                         float finfo;
                        } info;
                } mybuf;
```

Параметърът length трябва да съдържа максималната дължина на полезната част от информацията (т. е. информацията, разположена в структурата след неговия тип), която може да бъде извлечена от съобщението.
При успешно приключване, системното извикване копира избраното съобщение от опашката в паметта, чийто адрес е указанн от параметъра ptr, след което го изтрива от опашката.
Параметърът flag може да приема стойност 0 или която и да е комбнация от флагове IPC_NOWAIT и MSG_NOERROR. Ако флага IPC_NOWAIT не е установен и опашката е празна или в нея няма съобщения от указания тип, то системното извикване блокира до появата на заявеното съобщение. Ако се стигне до описаната ситуация и е установен флага IPC_NOWAIT, то системното извикване не блокира, а констатира възникване на грешка и в променливата errno, описанна във файла <errno.h> се записва стойност EAGAIN. Ако действителната дължина на полезната част от информацията в избраното съобщение превишава стойността, указана в параметъра length и флага MSG_NOERROR не е зададен, съобщение не се избира и се фиксира наличие на грешка. В този случай, ако флага MSG_NOERROR е

установен, грешка не възниква, а съобщението се копира в съкратен вид.

ВРЪЩАНИ СТОЙНОСТИ

Системното извикване връща действителната дължина на полезната част от информацията, копирана от опашката от съобщения (т. е. информацията, разположена в структурата след типа на съобщението) при нормално приключване и стойност -1 при възникване на грешка.

Максимално възможната дължина на информативната част от съобщението в операционната система Linux съставлява 4080 байта. Тя може да бъде намалена при компилиране на ядрото на системата. Текущата стойност на тази максимална дължина може да се определи с помощта на командата

```
ipcs -l
```

8. Изтриване на опашки от съобщения

След завършване на процесите, използващи дадена опашка от съобщения, тя не се премахва автоматично от системата, а продължава да се съхранява в нея, заедно с всички неизползвани съобщения. Неизползваните опашки от съощения се изтриват с помоща на специална команда или специално системно извикване. Синтаксиса на командата използвана за изтриване на опашка от съобщения е следния:

```
ipcrm msg <IPC идентификатор>
```

За получаване на IPC идентификатора на опашката от съобщения се използва командата *ipcs*.

Системното извикване за изтриване на опашка от съобщения е *msgctl()*. То може да се използва и за изпълнение на други операции с опашки от съобщения. За повече подробности вижте [UM].

Ако съществува процес, който е блокирал вследствие на записа или четенето в опашката от съобщения, то след изтриване на опашката процеса бива деблокиран.

ИМЕ

msgctl

ПРОТОТИП

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

ОПИСАНИЕ

Системното извикване msgctl е предназначено за получаване на информация от опашката от съобщения, промяна на нейните атрибути и изтриването и от системата. Даденото описание не е пълно и е адаптирано за настоящия курс. Повече информация може да се намери в UNIX Manual.

В настоящото упражнение системното извикване msgctl се използва единствено за изтриване на опашки от системата. Параметърт msqid е System V IPC дескриптора на изтриваната опашка от съобщения. Неговата стойност е тази върната от системнто извикване msgget() при създаване на опашката или при достъпа до нея по зададен ключ.
В настоящия курс на параметра cmd винаги се задава стойност IPC_RMID, която служи като команда за изтриване на опашката. Параметърът buf в този случай не е необходим и му се присвоява стойност NULL.

ВРЪЩАНИ СТОЙНОСТИ

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

За илюстрация на описаното са представени две програми:

```
/* Програма 05_1a.c, илюстрираща работата с опашки от съобщения */
/* Тази програма получава достъп до опашка от съобщения,
изпраща в нея 5 текстови съобщения от тип 1 и едно
празно съобщение от тип 255, което се използва като сигнал
към програмата 05_1b.c, за прекратяване на нейната работа */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <stdio.h>
#define LAST_MESSAGE 255 /* Тип на съобщението за
    прекратяване на работата на програма 05_1b.c  */
int main()
{
    int msqid; /* IPC дескриптора на опашката */
char pathname[] = "05_1a.c"; /* Име на файла, който се
        използва за генериране на ключ. Файл с такова
        име трябва да съществува в текущата директория */
    key_t key; /* IPC ключ */
    int i,len; /* Броячи: за цикъла и за дължината на
        информативната част от съобщението */
    /* Следва потребителската структура на
        съобщението */
    struct mymsgbuf
    {
        long mtype;
        char mtext[81];
    } mybuf;
    /* Генериране на IPC ключа от файловото име 05_1a.c в текущата
       директория и номера на инстанцията на опашката – 0. */
    if((key = ftok(pathname,0)) < 0){
        printf("Can\'t generate key\n");
        exit(-1);
    }
    /* Опит да се получи достъп до опашката по зададения ключ,
    ако тя съществува, или да бъде създадена, с права за достъп
    read & write за всички потребители */
    if((msqid = msgget(key, 0666 | IPC_CREAT)) < 0){
        printf("Can\'t get msqid\n");
        exit(-1);
    }
    /* Използване на цикъл за изпращане на 5 съобщения от тип 1
    в опашка от съобщения, с идентификатор msqid.*/
    for (i = 1; i <= 5; i++){
        /* Първоначално се запълва структурата на съобщението
           и се определя дължината на неговата информативна част */
        mybuf.mtype = 1;
        strcpy(mybuf.mtext, "This is text message");
        len = strlen(mybuf.mtext)+1;
        /* Съобщението се изпраща. В случай на грешка се съобщава
           за нея и опашката се изтрива от системата. */
        if (msgsnd(msqid, (struct msgbuf *) &mybuf,
            len, 0) < 0){
            printf("Can\'t send message to queue\n");
            msgctl(msqid, IPC_RMID,
                (struct msqid_ds *) NULL);
            exit(-1);
```

```
            }
        }
        /* Изпраща се съобщение от тип LAST_MESSAGE и дължина 0,
           което заставя приемащия процес да прекрати работата си. */
        mybuf.mtype = LAST_MESSAGE;
        len = 0;
        if (msgsnd(msqid, (struct msgbuf *) &mybuf,
            len, 0) < 0){
            printf("Can\'t send message to queue\n");
            msgctl(msqid, IPC_RMID,
                (struct msqid_ds *) NULL);
            exit(-1);
        }
        return 0;
}
```

Листинг 5.1а. Програма 1(05_1a.c), илюстрираща работата с опашки от съобщения

```
/* Програма 05_1b.c, илюстрираща работата с опашки от съобщения. */
/* Тази програма получава достъп до опашка от съобщения и чете от
   нея всички съобщения, без оглед на типа им в реда на тяхното
   постъпване, до момента в който не получи съобщение от тип 255,
   което е като сигнал за прекратяване на работата. */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <stdio.h>
#define LAST_MESSAGE 255 /* Тип на съобщението за
    прекратяване на работата */
int main()
{
    int msqid; /* IPC дескриптора на опашката */
    char pathname[] = "05_1a.c "; /* Име на файла, който се
        използва за генериране на ключ. Файл с такова
        име трябва да съществува в текущата директория */
    key_t key; /* IPC ключ */
    int len, maxlen; /* Реалната дължина и максималната
        дължина на информативната част от съобщението */
    /* Следва потребителската структура за съобщението */
    struct mymsgbuf
    {
        long mtype;
        char mtext[81];
    } mybuf;
    /* Генериране на IPC ключа от файловото име 05_1a.c  в текущата
       директория и номера на инстанцията на опашката – 0*/
    if((key = ftok(pathname,0)) < 0){
        printf("Can\'t generate key\n");
        exit(-1);
    }
    /* Опит да се получи достъп до опашката по зададения ключ,
       ако тя съществува, или да бъде създадена, с права за достъп
       read & write за всички потребители */
    if((msqid = msgget(key, 0666 | IPC_CREAT)) < 0){
        printf("Can\'t get msqid\n");
        exit(-1);
    }
    while(1){
        /* В безкраен цикъл се приемат съобщения от всеки тип,
           в реда на постъпването им с максимална дължина
           на информативната част 81 символа до момента, в който
           не постъпи съобщение от LAST_MESSAGE*/
        maxlen = 81;
        if(( len = msgrcv(msqid,
            (struct msgbuf *) &mybuf, maxlen, 0, 0) < 0){
            printf("Can\'t receive message from queue\n");
            exit(-1);
        }
        /* Ако бъде прието съобщение от тип LAST_MESSAGE,
           се прекратява работата и опашката се изтрива
```

```
                от системата. В противен случай се отпечатва
                текста на приетото съобщение. */
             if (mybuf.mtype == LAST_MESSAGE){
                 msgctl(msqid, IPC_RMID,
                 (struct msqid_ds *) NULL);
             exit(0);
             }
             printf("message type = %ld, info = %s\n",
             mybuf.mtype, mybuf.mtext);
        }
             return  0;  /*  Само  за  да  не  се  появяват  warning'и  при
                      компилацията. */
}
```

Листинг 5.1b. Програма 2 (05_1b.c), илюстрираща работата с опашки от съобщения

Първата от тези програми изпраща пет текстови съобщения от тип *1* и едно съобщение с нулева дължина от тип *255* към втората програма. Втората програма организира цикъл в който приема всички съобщения от всички типове по реда на постъпването им (организация от тип FIFO) и отпечатва тяхното съдържание. При получаване на съобщение от тип *255* програмата изтрива опашката от съобщения и прекратява работата си. Ако преди стартиране, на която и да е от програмите, опашката от съобщения липсва в системата, то програмите я създават.

Трябва да се отбележи, че в случая съобщението от тип 255 се използва в качеството му на сигнал за прекратяване на работата на втория процес. Това съобщение има нулева дължина, така че неговата информативност се изчерпва само с наличието му.

Задача 1: Въведете програмите, съхранете ги, съответно като файлове с имена *05_1a.c* и *05_1b.c* , компилирайте и проверете тяхното поведение.

Информацията предавана с помоща на системните извиквания *msgsnd()* и *msgrcv()* не е задължително да бъде само текстова. Опашките от съобщения могат да се използват за предаване на данни от всякакъв вид. При предаване на разнородна информация е целесъобразно да се използва отделна структура, която да обединява информативната част вътре в съобщението:

```
struct mymsgbuf {
    long mtype;
    struct {
        short sinfo;
        float finfo;
    } info;
} mybuf;
```

така че да може правилно да се изчисли нейния размер. В някой

изчислителни системи числовите данни се разполагат в паметта с подравняване по определени адреси (например, адреси кратни на 4). Ето защо размера на заетата памет може да се окаже по-голям от дължината на съдържащите се в нея числови данни, в горния случай:

```
sizeof(info)>=sizeof(short)+sizeof(float)
```

Дължината на информативната част от съобщението се указва не като сума от дължините на отделните полета в структурата, а като се вземе предвид дължината на цялата структура.

> *Задача 2:* Модифицирайте програмите *05_1a.c* и *05_1b.c* , така че да могат да се предават съобщения, съдържащи и числова информация.

Наличието на типове в разгледаните съобщения позволява да се организират двупосочни връзки между процеси, чрез използване на една и съща опашка. Даден процес може да изпраща на друг съобщения от един тип, а да получава от него съобщения от друг тип. При това получаването на съобщенията и в двата процеса следва да се извършва посредством втория от трите способа описани в т. 6 от упражнението.

> *Задача 3:* Напишете, компилирайте и изпълнете програма, осъществяваща двупосочна връзка, посредством една опашка от съобщения.

10. Мултиплексиране на съобщения. Модел клиент-сървър за взаимодействие на процеси. Неравнопоставеност на клиента и сървъра

Предходния пример показва, че е възможно да се организира мултиплексиране на съобщенията, т.е. един процес може да получава съобщения от множество други процеси, само, чрез една опашка. Същевременно, процесът връща отговори, използвайки същата опашка.

Принципно под мултиплексиране на информацията се разбира възможността едновременно да се обменя информация с няколко партньори. Мултиплексирането е широко използвано в модела клиент-сървър за взаимодействие на процеси. В този модел един от процесите е сървър. Сървърът получава заявки от други процеси (процеси-клиенти), обработва заявките, след което изпраща отговори с резултатите от обработката. Преди всичко, модела клиент-сървър се използва при

разработката на мрежови приложения. Едно от основните предположения в този модел е, че взаимодействащите си процеси са неравноправни:

•Като правило, сървърът работи постоянно, през цялото време на живот на приложението, а клиентът може да работи епизодично.

•Сървърът чака заявки от клиентите. Именно клиентите са инициатори на взаимодействията.

•Като правило, в даден момент клиентът се обръща само към един сървър, в същото време към сървърът могат да постъпят няколко клиентски заявки.

•Клиентът е длъжен, преди да организира заявка към сървъра, да знае как да се обръща към него (например, какъв е типа съобщенията възприемани от сървъра). В същото време, сървърът може да получи необходимата му информация за организиарне на взаимодействието с клиента от постъпилата заявка.

Следва описание на схема за мултиплексиране на съобщения, реализирана, чрез една опашка от съобщения по модела клиент-сървър: Сървър получава от опашка само съобщения от тип 1. Чрез тези съобщения процесите-клиенти изпращат към сървъра стойността на своя идентификатор на процеса. След като сървърът получи съобщение от тип 1, той анализира съдържанието му, извлича идентификатора на процеса, отправил заявката и отговаря на клиента, изпращайки съобщение с тип, равен на идентификатора на процеса направил заявката. След като отправи заявката процеса-клиент очаква отговор във вид на съобщение с тип равен на своя идентификатор. Тъй като идентификаторите на процесите в системата са различни и нито един потребителски процес не може да има PID равен на 1, процесите получават само съобщенията адресирани до тях. Ако обработката на заявките отнема продължително време, сървърът може да организира паралелна обработка на заявките, пораждайки за всяка заявка нов процес-дете, който да поеме нейното изпълнение.

Задача 4: Напишете и компилирайте две програми: клиент и сървър, реализиращи предложената схема на мултиплексиране. Стартирайте програмите и проверете работата им.