

KOLEJKI KOMUNIKATÓW I PC

Współbieżność procesów (czy wątków) wiąże się często z potrzebą okresowej (asynchronicznej) wymiany komunikatów. Żaden z omawianych wcześniej mechanizmów nie spełniłby się najlepiej w tej roli. Po raz pierwszy problem ten znalazł rozwiązanie w

UNIX System V

w postaci

kolejki komunikatów, ang. message queues

wprowadzonym jeszcze przez *AT&T* w roku 1983, a stosowanych do chwili obecnej w rodzinie systemów nawiązujących po spuścizny UNIX.

Kolejki komunikatów stanowią jeden z kilku elementów funkcjonalnych tworzących

InterProcess Communication, IPC

implementowany standardowo w obrębie jądra systemów będących potomkami UNIX System V.

Również *MsWindows*, którego twórcy mają od pewnego czasu aspiracje stworzenie systemu wspierającego współbieżność dysponuje także swoistym *IPC*.

KOLEJKI KOMUNIKATÓW IPC

Kolejka komunikatów stanowi połączona listę wiadomości przechowywanych w obszarze pamięci jądra systemowego i jest identyfikowana przez unikalny klucz – identyfikator **qmsqid** (*queue identifier*) – będący nieujemną liczbą całkowitą.

Z poziomu interface'u użytkownika informacja o istniejących kolejkach komunikatów może być pobrana poleceniem

```
$ ipcs -q
----- Message Queues -----
key          msqid      owner          perms          used-bytes   messages
0x0000000a 0          kmirota       777            0            0
$
```

Kolejka komunikatów może być usunięta z obszaru pamięci jądra poleceniem

```
$ ipcrm [-Q msgkey] | [-q msgid]
```

W pokazanym wcześniej wyniku polecenia **ipcs** byłoby to

```
$ ipcrm -Q 0x0000000a
```

lub

```
$ ipcrm -q 0
```

KOLEJKI KOMUNIKATÓW IPC

Celem obsługi tego sposobu komunikacji, w obszarze pamięci jądra tworzona jest tablica kolejek `msgque[]` o rozmiarze (maksymalnym) `MSGMNI`, której elementem składowym są struktury o definicji (w `linux/msg.h`)

```
struct msqid_ds
{
    struct ipc_perm    msg_perm;
    struct msg         *msg_first;    /*pierwszy komunikat w kolejce*/
    struct msg         *msg_last;    /*ostatni komunikat w kolejce*/
    __kernel_time_t    msg_stime;    /*czas ostatniego msgsnd*/
    __kernel_time_t    msg_rtime;    /*czas ostatniego msgrcv*/
    __kernel_time_t    msg_ctime;    /*czas ostatniej zmiany*/
    struct wait_queue  *wwait;
    struct wait_queue  *rwait;
    unsigned short     msg_cbytes;    /*liczba bajtów w kolejce*/
    unsigned short     msg_qnum;      /*liczba komunikatów w kolejce*/
    unsigned short     msg_qbytes;    /*maksymalna liczba bajtów w kolejce*/
    __kernel_ipc_pid_t msg_lspid;     /*pid ostatniego msgsnd() */
    __kernel_ipc_pid_t msg_lrpid;     /*pid ostatniego msgrcv() */
};
```

Tablica ta jest indeksowana za pośrednictwem identyfikatora danej kolejki `qid`. Zauważmy, że Struktura ta zawiera w szczególności wskazania, w postaci `msg_first` i `msg_last` identyfikujące listę komunikatów powiązanych z daną kolejką.

KOLEJKI KOMUNIKATÓW IPC

Nowa kolejka jest tworzona, co wiąże się z "dopisaniem" do tablicy `msgque` struktury `msqid_ds` a jeżeli już istnienie, to jest otwierana celem uzyskania dostępu wywołaniem funkcji `msgget()`.

SYNOPSIS

```
#include <sys/msg.h>
int msgget( key_t key, int flag );
key_t key predefiniowana stała IPC_PRIVATE (jeżeli kolejka prywatna)
albo unikalny klucz jeżeli ma być dostępna publicznie
(w pierwszym przypadku klucz wygeneruje jądro systemowe)
int flag zwykle IPC_CREAT polecająca utworzyć kolejkę
(o ile nie istnieje)
```

RETURN

`int` *indeks msqid do tablicy kolejek msgque.
identyfikujący w niej strukturę msqid_ds*

ERRORS

-1

KOLEJSKI KOMUNIKATÓW IPC

Składowe `msqid_ds` do której funkcja zwróciła indeks inicjowane są w taki sposób że:

- ❑ `msg_perm.cuid` i `msg_perm.uid` oraz odpowiednio `msg_perm.cgid` i `msg_perm.gid` inicjowane są wartościami identyfikatora właściciela procesu oraz jego grupy;
- ❑ na pozycję 9 najmniej znaczących bitów pola `msg_perm.mode` jest kopiowanych z 9 najmniej znaczących bitów wartości aktualnej parametru formalnego `msgflg`, przy czym `ipc_perm` jest zdefiniowane w ogólności jako

```
struct ipc_perm
{
    key_t __key;          /* key wysłany przez msgget() */
    uid_t uid;            /* UID właściciela */
    gid_t gid;            /* GID grupy właściciela */
    uid_t cuid;          /* UID twórcy */
    gid_t cgid;          /* GID grupy twórcy */
    unsigned short mode;  /* uprawnienia */
    unsigned short seq;   /* numer sekwencyjny */
};
```

- ❑ `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime` i `msg_rtime` inicjowane są zerami (0);
- ❑ `msg_ctime` będzie odpowiadać aktualnemu czasowi;
- ❑ `msg_qbytes` przyjmuje wartość równą ograniczeniu systemowemu `MSGMNB` (zdefiniowanej w `linux/msg.h`) jako

```
#define MSGMNB 16384 /* <=INT_MAX, default max size of a message queue*/
```

KOLEJSKI KOMUNIKATÓW IPC

Zauważmy, że dzięki przyjętemu sposobowi inicjowania pola `msg_perm.mode` za pośrednictwem argumentu `flag`, wykorzystując operator alternatywy bitowej `'|'` możemy określić w inny sposób – niż domniemany – uprawnienia

`flag = IPC_CREAT | mode`

jest to możliwe dzięki temu, że wg definicji stałej `IPC_CREAT` jej wartość wynosi 512, czyli binarnie

1 000 000 000

a więc właśnie 9 najmłodszych bitów jest nie wykorzystywanych.

Uprawnienia są określane, jak i w pozostałych przypadkach, za pomocą maski bitowej

<i>read by user</i>	(u+r)	00400	0x100	256
<i>write by user</i>	(u+w)	00200	0x80	128
<i>read by group</i>	(g+r)	00040	0x20	32
<i>write by group</i>	(g+w)	00020	0x10	16
<i>read by others</i>	(o+r)	00004	0x4	4
<i>write by others</i>	(o+r)	00002	0x2	2

KOLEJKI KOMUNIKATÓW IPC

Chociaż klucz key wywołaniu `msgget()` może być zadany w sposób dowolny, to jednak w przypadku kiedy będzie wykorzystywany już wcześniej, to funkcja zwróci błąd. W przypadku kiedy:

- ❑ ma być to kolejka prywatna albo użytkowana przez potomków utworzonych funkcją `fork()` (albo `fork()` i dalej `exec*()`) w zupełności wystarczającym jest podstawienie `IPC_PRIVATE`;

- ❑ jeżeli jednak ma kolejka ma być użytkowana przez procesy niespokrewnione, to wygodnie jest wygenerować wspólny dla wszystkich klucz wywołaniem funkcji `ftok()`.

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
key_t ftok( const char *pathname, int id );
const char *pathname  pewna arbitralnie ustalona ścieżka
int id  pewien arbitralnie ustalony identyfikator
 (liczba całkowita niezerowa)
```

RETURN

`key_t` *wygenerowany klucz, w przypadku powodzenia*

ERRORS

-1

Jako tako jej zastosowanie nie ogranicza się wyłącznie do identyfikacji kolejek.

OBLICZENIA RÓWNOLEGŁE I SYSTEMY ROZPROSZONE
KRYSPIŃ MIROTA, KMIROTA@ATH.BIELSKO.PL

KOLEJKI KOMUNIKATÓW IPC

Ogół działań związanych z zarządzaniem kolejką, do której proces posiada uprawnienia, realizuje funkcja `msgctl()`.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl( int msqid, int cmd, struct msqid_ds *buf );
int msqid identyfikator kolejki komunikatów w tablicy systemowej
int cmd rodzaj operacji do wykonanie
struct msqid_ds *buf wskazanie do struktury w której będzie zwracany wynik
albo z której będą pobierane wartości
```

RETURN

0 w przypadku sukcesu

ERRORS

-1

Na kolejce mogą być wykonane następujące operacje `cmd`:

IPC_STAT pobranie informacji z systemowej `msqid_ds`;

IPC_SET przekopiowanie wartości zadanych trzecim parametrem formalnym do systemowej struktury `msqid_ds`;

IPC_RMID natychmiastowe usunięcie kolejki komunikatów.

OBLICZENIA RÓWNOLEGŁE I SYSTEMY ROZPROSZONE
KRYSPIŃ MIROTA, KMIROTA@ATH.BIELSKO.PL

KOLEJKI KOMUNIKATÓW IPC

W implementacji *LINUX*owej uzyskujemy tu nieco większą funkcjonalność w relacji do *UNIX SysV* czy *POSIX*.

W szczególności określając

`cmd = IPC_INFO`

uzyskujemy ogólne informacje odnośnie realizacji i ograniczeń kolejek komunikatów w danym systemie. Wynik zwracany jest w strukturze `msginfo` (definicja `sys/msg.h`).

```
struct msginfo
{
    int msgpool; /* --- aktualnie nieużywana --- */
    int msgmap; /* --- aktualnie nieużywana --- */
    int msgmax; /* max ilość bajtów
                  jaka może być przesłana w pojedynczej wiadomości */
    int msgmnb; /* max ale ogółem do kolejki */
    int msgmni; /* max ilość kolejek */
    int msgssz; /* --- aktualnie nieużywana --- */
    int msgtql; /* --- aktualnie nieużywana --- */
    unsigned short int msgseg; /* --- aktualnie nieużywana --- */
};
```

KOLEJKI KOMUNIKATÓW IPC

Jako przykład przygotujemy program, który utworzy kolejkę, pobierze podstawowe informacje na jej temat.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/msg.h>
#include <time.h>
#define PROJECTID 666
```

```
int main( void )
{
```

```
    key_t key;
    int flag, msqid;
    struct msqid_ds buffer;
```

Generujemy klucz dla potrzeb utworzenia kolejki podając pewien arbitralnie obrany (ale – koniecznie – istniejący katalog; z pewnością `/tmp` będzie istniał), a ponadto pewną wartość liczbową (tutaj stała `PROJECTID`)

```
key = ftok( "/tmp", _PROJECT_ID );
```

Następnie tworzymy kolejkę (uprawnienia odczytu i zapisu tylko dla użytkownika)

```
flag = IPC_CREAT | 0x100 | 0x80;
```

```
msqid = msgget( key, flag );
```

OBLICZENIA RÓWNOLEGŁE I SYSTEMY ROZPROSZONE

KRYSPIŃ MIROTA, KMIROTA@ATH.BIELSKO.PL

KOLEJKI KOMUNIKATÓW IPC

Sprawdzamy czy faktycznie kolejka powstała,
czyli zostało umieszczone do niej wskazanie w tablicy systemowej

```
if( msqid<0 ){ perror( "!!...msgget().." ); exit( 1 ); }  
else  
{
```

Jeżeli tak, to pobieramy o niej informacje

```
if( !msgctl( msqid,IPC_STAT,&buffer ) )  
{
```

Identyfikator kolejki

```
printf( "\tKOLEJKA [%d]\n", (int)msqid );
```

Dla pewności pobieramy nasz UID i GID

```
printf( "\tuid:%d gid:%d\n", (int)getuid(), (int)getgid() );
```

... i sprawdzenia zawartości struct msqid_ds

```
printf( "\t%s", ctime( &(buffer.msg_ctime) ) );
```

```
printf( "\tuid:%dgid:%d\n", (int)buffer.msg_perm.uid,  
        (int)buffer.msg_perm.gid );
```

```
printf( "\tkey: 0x%x\n", buffer.msg_perm.__key );
```

```
printf( "\trozmiar kolejki: %lu B / %u MSG\n",  
        buffer.msg_cbytes, (unsigned)buffer.msg_qnum );
```

```
/* msgctl( msqid,IPC_RMID,&buffer ); */
```

```
}else{ perror( "!!...msgctl.." ); exit( 2 ); }
```

```
}
```

```
return 0;
```

```
}
```

OBLICZENIA RÓWNOLEGŁE I SYSTEMY ROZPROSZONE
KRYSPIŃ MIROTA, KMIROTA@ATH.BIELSKO.PL

KOLEJSKI KOMUNIKATÓW IPC

Jeżeli program zapiszemy pod nazwą msgtest.c, to jego kompilacja i konsolidacja

```
$ gcc -Wall msgtest.c -o msgtest
```

Sprawdźmy jeszcze przed uruchomieniem programu listę kolejek w pamięci jądra

```
# ipcs -q
```

```
----- Message Queues -----
```

key	msqid	owner	perms	used-bytes	messages
-----	-------	-------	-------	------------	----------

czyli brak kolejek. Uruchommy teraz program

```
$ ./msgtest
```

```
KOLEJKA [32768]
```

```
uid:1000 gid:100
```

```
Wed Oct 21 09:42:40 2008
```

```
uid:1000 gid:100
```

```
key: 0x9a060481
```

```
rozmiar kolejki: 0 B / 0 MSG
```

Mamy zatem kolejkę :

- ☐ identyfikowaną przez msqid=32768
- ☐ klucz key=0x9a060481
- ☐ utworzona msg_ctime=Wed Oct 21 09:42:40 2008
- ☐ właściciela opisuje uid:1000 gid:100, czyli user=kmirota group=users
- ☐ w kolejce jest aktualnie 0 wiadomości.

OBLICZENIA RÓWNOLEGŁE I SYSTEMY ROZPROSZONE
KRYSPIŃ MIROTA, KMIROTA@ATH.BIELSKO.PL

KOLEJKI KOMUNIKATÓW IPC

Sprawdźmy raz jeszcze już po zakończeniu procesu listę istniejących kolejek

```
$ ipcs -q
----- Message Queues -----
key          msqid          owner          perms          used-bytes      messages
0x9a060481 32768          kmirota        600             0                0
```

czyli, mimo że proces już nie istnieje kolejka pozostała w pamięci. Co nie mniej istotne, nawet gdyby zamknąć sesję terminala i ponownie otworzyć, to

```
$ ipcs -q
----- Message Queues -----
key          msqid          owner          perms          used-bytes      messages
0x9a060481 32768          kmirota        600             0                0
```

a więc kolejki nie są usuwane automatycznie z pamięci, a wymagają aby wykonał to jawnie proces który kolejką utworzył. Warto pamiętać o tym fakcie, a z naszego kodu należy zmodyfikować wprowadzając zamiast

```
...
/* msgctl(    msqid,IPC_RMID,&buffer ); */
}else{ perror( "!!...msgctl.." ); exit( 2 ); }
```

powinno być

```
...
msgctl(  msqid,IPC_RMID,&buffer );
}else{ perror( "!!...msgctl.." ); exit( 2 ); }
```

czyli usunąć komentarz. Dopiero teraz kolejka zostanie usunięta z pamięci jądra.

OBLICZENIA RÓWNOLEGŁE I SYSTEMY ROZPROSZONE
KRYSPIŃ MIROTA, KMIROTA@ATH.BIELSKO.PL

KOLEJKI KOMUNIKATÓW IPC

Przesyłanie do i pobierane z kolejki wywołaniami funkcji `msgsnd()` i `msgrcv()`.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd( int msqid, const void *msgp, size_t msgsz, int msgflg );
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long mtype, int msgflg);
int msqid identyfikator kolejki do której wysyłana jest wiadomość
void *msgp wskazanie do struktury definiowanej przez wywołującego (!)
struct msgbuf
{
    long mtype;    /* message type, must be > 0 */
    char mtext[msgsz]; /* message data */
};
opisującej typ wiadomości i samą wiadomość
size_t msgsz rozmiar łańcucha mtext w strukturze msgp
(nie więcej niż MSGMAX=8192, zdefiniowane w linux/msg.h)
int msgflg jeżeli IPC_NOWAIT to nie czeka na możliwość zapisu lub odczytu
long mtype liczna identyfikująca rodzaj komunikatu na liście,
służąca ich filtrowaniu
```

RETURN

0 sukces `msgsnd()`, a `msgrcv()` ilość bajtów skopiowanych do tablicy `mtext`

ERRORS

-1

OBLICZENIA RÓWNOLEGŁE I SYSTEMY ROZPROSZONE
KRYSPIŃ MIROTA, KMIROTA@ATH.BIELSKO.PL

KOLEJKI KOMUNIKATÓW IPC

W wyniku pomyślnego wykonania operacji `msgsnd()`:

- ☐ `msg_lspid` będzie ustawione na **ID** procesu, który wysłał
- ☐ `msg_qnum` będzie zwiększone o 1
- ☐ `msg_stime` uzyska wartość bieżącej chwili czasu

natomiast `msgrcv()`:

- ☐ zapisanie kopi komunikatu wskazywanego przez `msgp` do kolejki o identyfikatorze `msqid` oraz jego usunięcie z kolejki systemowej
- ☐ parameter `msgsz` określa tutaj maksymalna długość łańcucha komunikatu, jeżeli w rzeczywistości długość okaże się większa, to wynik zależy od czego czy w `msgflg` podano **MSG_NOERROR** (tak – łańcuch będzie obcięty, nie – będzie powstanie błąd, czyli zwróci -1)

Parametr określający typ wiadomości `mtype`, który umożliwiając ich filtrowanie na etapie odczytu. Jego użycie w `msgrcv()` jest następujące:

- ☐ `mtype == 0`, czytana jest pierwsza dostępna wiadomość w kolejce
- ☐ `mtype > 0`, pierwsza wiadomość w kolejce o identycznej wartości typu (podany w momencie wysyłania jako parametr `msgsnd()`), jeżeli jednak **MSG_EXCEPT** podano w `msgflg` to odczytana będzie pierwsza i typie różnym od podanego w `mtype`;
- ☐ `mtype < 0`, pierwsza wiadomość o typie, w wywołaniu `msgsnd()`, $\leq |mtype|$.

KOLEJKI KOMUNIKATÓW IPC

W kolejnym przykładzie proces utworzy prywatną kolejkę, wyśle do niej dwa komunikaty a następnie spróbuje go odebrać.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/msg.h>
#define MSGMAX 8192
#define PROJECTID 0x1A

int main( void )
{
    key_t key;
    unsigned msqid,pid;
    Definiujemy struktury do obsługi kolejki i przesyłania komunikatów
    struct msqid_ds buffer;
    struct { long type; char text[MSGMAX]; } message;
    I w końcu inicjujemy kolejkę komunikatów generując klucz przez ftok()
    key = ftok( "/tmp",PROJECTID );
    msqid = msgget( key,IPC_CREAT|0x100|0x80 );
```

KOLEJKI KOMUNIKATÓW IPC

Jeżeli kolejka msqid zainicjowana pomyślnie, to wysyłamy do niej komunikaty

```
if( msqid<0 ){ perror( "!!!..msgget().." ); exit( 1 ); }  
else  
{
```

Niech komunikaty będą identyfikowane przez PID procesu wysyłającego

```
pid = getpid();
```

```
printf( "[%u] utworzył kolejkę...%u\n",pid,msqid );
```

```
message.type = (long)pid;
```

Przygotowujemy i wysyłamy pierwszą wiadomość

```
(void) strcpy( message.text, "'<1> pierwsza wiadomość <1>'");
```

```
printf( "[%u] wysła...\t%s\n",pid,message.text );
```

Konstrukcja `strlen(message.text)+1` konieczna, ponieważ `msgsnd()` wymaga podania ilości bajtów, a `strlen()` zwraca długość łańcucha bez końcowego `'\0'`

```
msgsnd( msqid,(void*)&message,strlen(message.text)+1,IPC_NOWAIT );
```

... no i druga wiadomość

```
(void) strcpy( message.text, "'<2> druga wiadomość <2>'");
```

```
printf( "[%u] wysła...\t%s\n",pid,message.text );
```

```
msgsnd( msqid,(void*)&message,strlen(message.text)+1,IPC_NOWAIT );
```

Teraz zajmiemy się odbiorem wysłanych komunikatów.

KOLEJKI KOMUNIKATÓW IPC

Generalnie lepiej pamiętać o ...

```
bzero( (void*)&message,sizeof( message ) );
```

Dopóki jeszcze coś jest w kolejce identyfikowanego typem `pid`, to czytamy

```
while( msgrcv( msqid,(void*)&message,MSGMAX,  
    pid,IPC_NOWAIT|MSG_NOERROR ) > 0 )  
{  
    printf( "[%u] otrzymał...\t%s\n",pid,message.text );  
    bzero( (void*)&message,sizeof( message ) );  
}
```

Skoro nic nie pozostało, to usuwamy z pamięci naszą kolejkę

```
msgctl( msqid,IPC_RMID,&buffer );
```

```
}
```

i kończymy

```
return 0;
```

```
}
```