

Kolejki komunikatów - materiały pomocnicze

Mechanizmy IPC

- Podobnie jak łącza, mechanizmy IPC (Inter Process Communication) jest grupą mechanizmów komunikacji i synchronizacji procesów działających w ramach tego samego systemu operacyjnego.

Mechanizmy IPC obejmują:

- kolejki komunikatów — umożliwiają przekazywanie określonych porcji danych,
- pamięć współdzieloną — umożliwiają współdzielenie kilku procesom tego samego fragmentu wirtualnej przestrzeni adresowej,
- semaforey — umożliwiają synchronizację procesów w dostępie do współdzielonych zasobów (np. do pamięci współdzielonej)

SYSTEM V

Wprowadzenie

- Kolejki komunikatów to specjalne listy (kolejki) w jądrze, zawierające odpowiednio sformatowane dane i umożliwiające ich wymianę poprzez dowolne procesy w systemie.
- Istnieje możliwość umieszczania komunikatów w określonych kolejkach (z zachowaniem kolejności ich wysyłania przez procesy) oraz odbierania komunikatu na parę różnych sposobów (zależnie od typu, czasu przybycia itp.).
- W systemie V kolejki komunikatów reprezentowane są przez strukturę `msqid_ds`.
- Do utworzenia obiektu potrzebny jest unikalny klucz w postaci 32-bitowej liczby całkowitej.

Klucz ten stanowi nazwę obiektu, która jednoznacznie go identyfikuje i pozwala procesom uzyskać dostęp do utworzonego obiektu.

Każdy obiekt otrzymuje również swój identyfikator, ale jest on unikalny tylko w ramach jednego mechanizmu.

Oznacza to, że może istnieć kolejka i zbiór semaforów o tym samym identyfikatorze.

Wartość klucza można ustawić dowolnie. Zalecane jest jednak używanie funkcji `ftok()` do generowania wartości kluczy.

Nie gwarantuje ona wprowadzenia unikalności klucza, ale znacząco zwiększa takie prawdopodobieństwo.

```
key_t ftok(char *pathname, char proj);
```

gdzie:

```
pathname - nazwa ścieżkowa pliku,  
proj - jednoliterowy identyfikator projektu.
```

- Wszystkie tworzone obiekty IPC mają ustalone prawa dostępu na podobnych zasadach jak w przypadku plików.
- Prawa te ustawiane są w strukturze `ipc_perm` niezależnie dla każdego obiektu IPC.

Obiekty IPC pozostają w pamięci jądra systemu do momentu, gdy:

- jeden z procesów zleci jądru usunięcie obiektu z pamięci,
- nastąpi zamknięcie systemu.

Polecenia systemowe

- Polecenie `ipcs` wyświetla informacje o wszystkich obiektach IPC istniejących w systemie, dokonując przy tym podziału na poszczególne mechanizmy.
- Wyświetlane informacje obejmują m.in. klucz, identyfikator obiektu, nazwę właściciela, prawa dostępu.

```
ipcs [ -asmq ] [ -tclup ]  
ipcs [ -smq ] -i id
```

Wybór konkretnego mechanizmu umożliwiają opcje:

```
-s - semafor,  
-m - pamięć dzielona,  
-q - kolejki komunikatów,  
-a - wszystkie mechanizmy (ustawienie domyślne).
```

Dodatkowo można podać identyfikator pojedynczego obiektu -i id, aby otrzymać informacje tylko o nim.

Pozostałe opcje specyfikują format wyświetlanych informacji.

- Dowolny obiekt IPC można usunąć posługując się poleceniem:

```
ipcrm [ shm | msg | sem ] id
```

gdzie:

```
shm, msg, sem - specyfikacja mechanizmu, kolejno: pamięć dzielona  
id - identyfikator obiektu.
```

Struktury danych

- Za każdą kolejkę komunikatów odpowiada jedna struktura typu msqid_ds.
- Komunikaty danej kolejki przechowywane są na liście, której elementami są struktury typu msg - każda z nich posiada informacje o typie komunikatu, wskaźnik do następnej struktury msg oraz wskaźnik do miejsca w pamięci, gdzie przechowywana jest właściwa treść komunikatu.
- Dodatkowo, każdej kolejce komunikatów przydziela się dwie kolejki typu wait_queue,

na których śpią procesy zawieszone podczas wykonywania operacji czytania bądź pisania do danej kolejki.

- Poniższy rysunek przedstawia wyżej omówione zależności:

(rysunek)

- W pliku `include/linux/msg.h` zdefiniowane są ograniczenia na liczbę i wielkość kolejek oraz komunikatów w nich umieszczanych:

```
#define MSGMNI 128      /* <= 1K      max # kolejek komunikatow
#define MSGMAX 4056     /* <= 4056   max rozmiar komunikatu (w bajt
#define MSGMNB 16384    /* ?           max wielkosc kolejki (w bajtach
```

Struktura `msqid_ds`

Dokładna definicja struktury `msqid_ds` z pliku `include/linux/msg.h`:

```
/* jedna struktura msg dla kazdej kolejki w systemie */
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 */
    __kernel_ipc_pid_t  msg_lspid;     /* pid ostatniego msgsnd */
    __kernel_ipc_pid_t  msg_lrpid;     /* pid ostatniego receive */
};
```

Dodatkowe wyjaśnienia:

`msg_perm`

- Jest to instancja struktury `ipc_perm`, zdefiniowanej w pliku `linux/ipc.h`.
- Zawiera informacje o prawach dostępu do danej kolejki oraz o jej założycielu.

wwait, rwait

- Przydzielone danej kolejce komunikatów dwie kolejki typu wait_queue, na których spią procesy zawieszone podczas wykonywania operacji odpowiednio czytania oraz pisania w danej kolejce komunikatów.

Struktura msg

- Dokładna definicja struktury msg z pliku include/linux/msg.h:

```
/* jedna struktura msg dla kazdego komunikatu */
struct msg {
    struct msg *msg_next; /* nastepny komunikat w kolejce */
    long        msg_type;
    char        *msg_spot;
    time_t      msg_stime; /* czas wyslania tego komunikatu */
    short       msg_ts;    /* dlugosc wlasciwej tresci komunikatu */
};
```

Dodatkowe wyjaśnienia:

msg_type

- Typ przechowywanego komunikatu.
- Wysyłanemu do kolejki komunikatowi nadawca przypisuje dodatnią liczbę naturalną, stającą się jego typem.
- Przy odbiorze komunikatu można zażądać komunikatów określonego typu (patrz opis funkcji msgrcv()).

msg_spot

- Wskaźnik do miejsca w pamięci, gdzie przechowywana jest właściwa treść komunikatu.

- Na każdy komunikat przydzielane jest oddzielne miejsce w pamięci.

Funkcje i ich implementacja

Istnieją cztery funkcje systemowe do obsługi komunikatów:

```
msgget() uzyskanie identyfikatora kolejki komunikatów używanego
msgctl() ustawianie i pobieranie wartości parametrów związanych
msgsnd() wysłanie komunikatu,
msgrcv() odebranie komunikatu.
```

Funkcja msgget()

- Funkcja służy do utworzenia nowej kolejki komunikatów lub uzyskania dostępu do istniejącej kolejki.

```
DEFINICJA: int msgget(key_t key, int msgflg)
WYNIK: identyfikator kolejki w przypadku sukcesu
      -1, gdy błąd: errno = EACCESS (brak praw)
                        EEXIST  (kolejka o podanym kluczu
                                już istnieje, więc niemożliwe jest j
                                jego utworzenie)
                        EIDRM   (kolejka została w międz
                                czasie usunięta)
                        ENOENT  (kolejka nie istnieje),
                        EIDRM   (kolejka została w międz
                                czasie usunięta)
                        ENOMEM  (brak pamięci na kolejki)
                        ENOSPC  (liczba kolejek w systemie
                                osiągnęła maksymalną)
```

- Pierwszym argumentem funkcji jest wartość klucza, porównywana z istniejącymi wartościami kluczy.
- Zwracana jest kolejka o podanym kluczu, przy czym flaga `IPC_CREAT` powoduje utworzenie kolejki w przypadku braku kolejki o podanym kluczu, zaś flaga `IPC_EXCL` użyta z `IPC_CREAT` powoduje błąd `EEXIST`, jeśli kolejka o podanym kluczu już istnieje.
- Wartość klucza równa `IPC_PRIVATE` zawsze powoduje utworzenie nowej kolejki.

- W przypadku konieczności utworzenia nowej kolejki, alokowana jest nowa struktura typu `msqid_ds`.

Funkcja `msgsnd()`

- Wysłanie komunikatu do kolejki.

```
DEFINICJA: int msgsnd(int msqid, struct msgbuf *msgp, int msgsz,
                    int msgflg)
WYNIK: 0 w przypadku sukcesu
      -1, gdy błąd: errno = EAGAIN (pełna kolejka (IPC_NOWAIT))
                        EACCES (brak praw zapisu)
                        EFAULT (zły adres msgp)
                        EIDRM (kolejka została w międz.)
                        EINTR (otrzymano sygnał podczas)
                        EINVAL (zły identyfikator kolejki)
                        ENOMEM (brak pamięci na komunikat)
```

- Pierwszym argumentem funkcji jest identyfikator kolejki.
- `msgp` jest wskaźnikiem do struktury typu `msgbuf`, zawierającej wysyłany komunikat.

Struktura ta jest zdefiniowana w pliku `linux/msg.h` następująco:

```
/* message buffer for msgsnd and msgrcv calls */
struct msgbuf {
    long mtype;          /* typ komunikatu */
    char mtext[1];       /* treść komunikatu */
};
```

- Jest to jedynie przykładowa postać tej struktury; programista może zdefiniować sobie a następnie wysłać dowolną inną strukturę, pod warunkiem, że jej pierwszym polem będzie wartość typu `long`, zaś rozmiar nie będzie przekraczał wartości `MSGMAX` (=4096).
- Wartość `msgsz` w wywołaniu funkcji `msgsnd` jest równa rozmiarowi komunikatu (w bajtach), nie licząc typu komunikatu (`sizeof(long)`).

- Flaga `IPC_NOWAIT` zapewnia, że w przypadku braku miejsca w kolejce funkcja natychmiast zwróci błąd `EAGAIN`.

Funkcja `msgrcv()`

- Odebranie komunikatu z kolejki.

DEFINICJA: `int msgrcv(int msgqid, struct msgbuf *msgp, int msgsz, long type, int msgflg)`

WYNIK: liczba bajtów skopiowanych do bufora w przypadku sukcesu, -1, gdy błąd: `errno = E2BIG` (długość komunikatu większa niż `msgsz`), `EACCES` (brak praw odczytu), `EFAULT` (zły adres `msgp`), `EIDRM` (kolejka została w międzyczasie usunięta), `EINTR` (otrzymano sygnał podczas oczekiwania), `EINVAL` (zły identyfikator kolejki), `ENOMSG` (brak komunikatu (IPC_NO

- Pierwszym argumentem funkcji jest identyfikator kolejki. `msgp` wskazuje na adres bufora, do którego ma być przekopiowany odbierany komunikat.
- `msgsz` to rozmiar owego bufora, z wyłączeniem pola `mtype` (`sizeof(long)`).
- `mtype` wskazuje na rodzaj komunikatu, który chcemy odebrać.

Jądro przydzieli nam najstarszy komunikat zadanego typu, przy czym:

- jeśli `mtype = 0`, to otrzymamy najstarszy komunikat w kolejce
- jeśli `mtype > 0`, to otrzymamy komunikat odpowiedniego typu
- jeśli `mtype < 0`, to otrzymamy komunikat najmniejszego typu mniejszego niż 0
- jeśli `msgflg` jest ustawiona na `MSG_EXCEPT`, to otrzymamy dowolny komunikat z wyjątkiem

- Ponadto, flaga `IPC_NOWAIT` w przypadku braku odpowiedniego komunikatu powoduje natychmiastowe wyjście z błędem, zaś `MSG_NOERROR` powoduje brak błędu w przypadku, gdy komunikat nie mieści się w buforze (zostaje przekopiowane tyle, ile się mieści).

Funkcja `msgctl()`

- modyfikowanie oraz odczyt rozmaitych właściwości kolejki.


```
DEFINICJA: int msgctl(int msgqid, int cmd, struct msqid_ds *buf)
WYNIK: 0 w przypadku sukcesu
      -1, gdy blad: errno = EACCES (brak praw czytania (IPC_S
                                EFAULT (zly adres buf)
                                EIDRM (kolejka zostala w miedz
                                EINVAL (zly identyfikator kolej
                                EPERM (brak praw zapisu (IPC_S
```

Dopuszczalne komendy to:

```
IPC_STAT: uzyskanie struktury msqid_ds odpowiadajacej kolejce (z
IPC_SET: modyfikacja wartosci struktury ipc_perm odpowiadajacej
IPC_RMID: skasowanie kolejki
```

- Działanie funkcji sprowadza się do przekopiowania odpowiednich wartości od lub do użytkownika, lub skasowania kolejki.

Usuniecie kolejki wygląda następująco:

```
{
    msqid_ds.ipc_perm.seq+=1;      /* patrz opis struktury ipc_per
                                   o cechach wspolnych mechaniz
    msg_seq+=1;                    /* zwiekszenie wartosci globalnej zmiennej
                                   ipc_perm.seq - patrz tenze rozdzial */
    uaktualnienie statystyk;
    msgque[msgqid]=IPC_UNUSED;
    obudzenie czekajacych na pisanie lub czytanie do/z usuwanej k
    zwolnienie struktur przydzielonych kolejce;
}
```

POSIX

POSIX (ang. Portable Operating System Interface for UNIX) - przenośny interfejs dla systemu UNIX.

- Jest to zestaw standardów opracowany przez stowarzyszenie IEEE (Institute of Electrical and Electronics Engineers) w 1985 roku w celu zapewnienia kompatybilności pomiędzy różnymi wersjami i dystrybucjami systemów operacyjnych.

- Standard ten definiuje zarówno interfejs programistyczny (API), jak i powłokę systemową oraz interfejs użytkownika.

Kolejki komunikatów

- Służą do wymiany komunikatów (ciągu danych o ustalonej długości i priorytecie) pomiędzy procesami.
- Kolejka to tak naprawdę lista, z której w czasie odczytu pobieramy najstarszy komunikat o najwyższym priorytecie (wg standardu POSIX).
- Pojedynczy komunikat zawiera priorytet (unsigned int), długość (size_t) oraz same dane, o ile długość jest większa niż 0 (char*).

Kolejki komunikatów tworzone są w określonym katalogu na dysku, np /DEV/mqueue.

Funkcje do obsługi kolejek komunikatów

Plik nagłówkowy

```
#include <mqueue.h>
```

Struktura struct mq_attr

```
struct mq_attr {  
    long mq_flags;    /* sygnalizator kolejki: 0, O_NONBLOCK */  
    long mq_maxmsg;   /* maksymalna liczba komunikatów w kolejce */  
    long mq_msgsize;  /* maksymalny rozmiar komunikatu (w bajtach) */  
    long mq_curmsgs;  /* liczba komunikatów w kolejce */  
};
```

Otwieranie kolejki

```
mqd_t mq_open(const char *name, int oflag [, mode_t mode, struct
```

- Funkcja ta próbuje otworzyć kolejkę komunikatów (która tak naprawdę jest plikiem o nazwie name).
- Zwraca deskryptor kolejki, jeśli się powiedzie lub -1 w przypadku błędu.
- Uwaga! Nazwa musi zaczynać się od znaku /

- Parametr oflag ma analogiczne znaczenie, jak w przypadku otwierania plików (unixowymi metodami obsługi plików).

Zatem akceptuje jedną z wartości: O_RDONLY, O_WRONLY, O_RDWR, kt O_CREAT, O_EXCL oraz O_NONBLOCK (aby używać tych stałych należy

- Parametr mode specyfikujemy, gdy tworzymy nową kolejkę i określa on prawa dostępu do niej.

Możemy podać wartość ósemkowo lub dowolną sumę logiczną stałych: (aby korzystać z tych stałych należy dołączyć plik nagłówkowy sy

- Ostatnim parametrem jest attr. Jest to struktura określająca parametry kolejki. Jeśli nie podamy tego parametru lub podamy NULL, to ustawione zostaną parametry domyślne.

Zamykanie kolejki

```
int mq_close(mqd_t mqdes);
```

- Funkcja ta zamyka kolejkę o deskrytorze mqdes.
- Zwraca 0 w przypadku sukcesu lub -1 w przypadku błędu.

Warto zauważyć, że funkcja ta nie niszczy kolejki, która dalej jest dostępna w systemie operacyjnym, ale jedynie ją zamyka.

- Gdy proces się kończy, automatycznie zamykane są wszystkie jego kolejki.

Usuwanie kolejki

```
int mq_unlink(const char *name);
```

- Usuwa z systemu kolejkę o nazwie name.
- Zwraca 0 w przypadku sukcesu lub -1 w przypadku błędu.
- Kolejka zostanie usunięta dopiero po zamknięciu jej przez wszystkie podłączone procesy.

Odczytywanie parametrów kolejki

```
int mq_getattr(mqd_t mqdes, struct mq_attr *attr);
```

- Odczytuje parametry kolejki o deskrytorze mqdes i zapisuje je w miejscu wskazywanym przez attr.
- Zwraca 0 w przypadku sukcesu lub -1 w przypadku błędu.

Ustawianie parametrów kolejki

```
int mq_setattr(mqd_t mqdes, const struct mq_attr *attr, struct m
```

- Ustawia parametry kolejki o deskrytorze mqdes wskazywane przez attr.
- Jeśli oattr nie wskazuje na NULL, to zapisywane są w tym miejscu stare parametry kolejki.
- Zwraca 0 w przypadku sukcesu lub -1 w przypadku błędu.

Wysyłanie komunikatów

```
int mq_send(mqd_t mqdes, const char* ptr, size_t len, unsigned i
```

- Wysyła komunikat wskazywany przez ptr do kolejki o deskrytorze mqdes o długości len i priorytecie prio.
- Zwraca 0 w przypadku powodzenia lub -1 w przypadku błędu.
- Priorytet nie może przekraczać MQ_PRIO_MAX!

Odbieranie komunikatów

```
ssize_t mq_receive(mqd_t mqdes, char *ptr, size_t len, unsigned
```

- Odbiera komunikat z kolejki o deskrytorze mqdes o długości len (co najmniej tyle, ile w polu mq_msgsize w strukturze struct mq_attr).
- Dane zapisuje do ptr, a priorytet do priop (o ile priop nie jest NULL).

- Zwraca liczbę odczytanych bajtów w przypadku powodzenia lub -1 w przypadku błędu.

Mechanizm powiadomień (ang. notifications)

- Mechanizm powiadomień pozwala na asynchroniczne zawiadamianie procesu, że w pustej kolejce umieszczono komunikat.

Może to się odbyć poprzez:

- wysłanie sygnału
- utworzenie wątku w celu wykonania określonej funkcji

Korzystanie z mechanizmu powiadomień

```
int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

- Funkcja ta powoduje zarejestrowanie (gdy notification nie jest równe NULL) lub wyrejestrowanie (gdy notification jest NULL) mechanizmu powiadomień dla kolejki o deskrytorze mqdes.
- Zwraca 0 w przypadku powodzenia lub -1 w przypadku błędu.

Struktura struct sigevent wygląda następująco:

```
struct sigevent {
    int      sigev_notify; /* sygnał czy wątek: SIGEV_NONE,
    int      sigev_signo;  /* numer sygnału (dla SIGEV_SIGN
    union sigval sigev_value; /* przekazywane procedurze obsłu
    /* dla SIGEV_THREAD występują jeszcze: */
    void (*sigev_notify_function)(union sigval);
    pthread_attr_t *sigev_notify_attributes;
};

union sigval {
    int    sival_int; /* wartość całkowitoliczbowa */
```

```
void *sival_ptr; /* wskaźnik */  
};
```

Korzystając z mechanizmu powiadomień należy pamiętać, że:

- W jednym procesie możemy korzystać z powiadomień tylko z jednej kolejki.
- Rejestracja obowiązuje tylko na jedno powiadomienie.
- Po powiadomieniu trzeba zarejestrować się ponownie, gdyż rejestracja jest kasowana.
- Jeśli w pustej kolejce pojawi się komunikat, a jednocześnie proces oczekuje na rezultat funkcji `mq_receive`, to do procesu nie zostanie wysłane powiadomienie. Nie ma to większego sensu, gdyż proces i tak oczekuje na wiadomość.