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,
* semafory — umożliwiają synchronizacje procesów w dostępie do współdzielonych zasobów (np. do pamięci współdzielonej)

SYSTEM V

Wprowadzenie

Kolejki komunikatów to specjalne listy (kolejki) w jadrze, 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 wprawdzie 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 ustalane 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** | - | semafory, |
| **-m** | - | pamięć dzielona, |
| **-q** | - | kolejki komunikatów, |
| **-a** | - | wszystkie mechanizmy (ustawienie domyślne). |

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

Pozostale opcje specyfikują format wyświetlanych informacji.

Dowolny obiekt IPC można usunąć posługując się poleceniem:

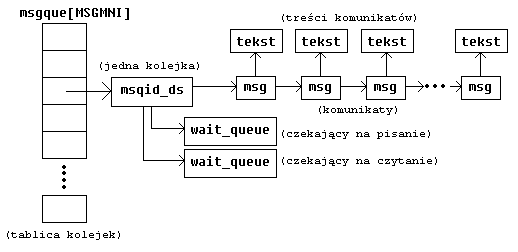
ipcrm [ shm | msg | sem ] id

gdzie:

|  |  |  |
| --- | --- | --- |
| **shm, msg, sem** | - | specyfikacja mechanizmu, kolejno: pamięć dzielona, kolejka komunikatów, semafory, |
| **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\_queque, na których śpią procesy zawieszone podczas wykonywania operacji czytania bądź pisania do danej kolejki. Poniższy rysunek przedstawia wyżej omówione zależności:



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 bajtach) \*/

#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 bajtow w kolejce \*/

unsigned short msg\_qnum; /\* liczba komunikatow w kolejce \*/

unsigned short msg\_qbytes; /\* maksymalna liczba bajtow w kolejce \*/

\_\_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

Dokladna 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 przez pozostałe funkcje,

msgctl() ustawianie i pobieranie wartości parametrów związanych z kolejkami komunikatów oraz usuwanie kolejek,

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 blad: errno = EACCESS (brak praw)

EEXIST (kolejka o podanym kluczu istnieje,

wiec niemozliwe jest jej utworzenie)

EIDRM (kolejka zostala w miedzyczasie skasowana)

ENOENT (kolejka nie istnieje),

EIDRM (kolejka zostala w miedzyczasie skasowana)

ENOMEM (brak pamieci na kolejke)

ENOSPC (liczba kolejek w systemie jest rowna

maksymalnej)

Pierwszym argumentem funkcji jest wartość klucza, porównywana z istniejacymi 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 blad: errno = EAGAIN (pelna kolejka (IPC\_NOWAIT))

EACCES (brak praw zapisu)

EFAULT (zly adres msgp)

EIDRM (kolejka zostala w miedzyczasie skasowana)

EINTR (otrzymano sygnal podczas czekania)

EINVAL (zly identyfikator kolejki, typ lub rozmiar

komunikatu)

ENOMEM (brak pamieci 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 nastepująco:

/\* message buffer for msgsnd and msgrcv calls \*/

struct msgbuf {

long mtype; /\* typ komunikatu \*/

char mtext[1]; /\* tresc komunikatu \*/

};

Jest to jedynie przykładowa postać tej struktury; programista może zdefiniować sobie a następnie wysyłać dowolną inną strukturę, pod warunkiem, że jej pierwszym polem będzie wartość typu long, zaś rozmiar nie będzie przekraczać 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 bajtow skopiowanych do bufora w przypadku sukcesu

-1, gdy blad: errno = E2BIG (dlugosc komunikatu wieksza od msgsz)

EACCES (brak praw odczytu)

EFAULT (zly adres msgp)

EIDRM (kolejka zostala w miedzyczasie skasowana)

EINTR (otrzymano sygnal podczas czekania)

EINVAL (zly identyfikator kolejki lub msgsz < 0)

ENOMSG (brak komunikatu (IPC\_NOWAIT))

Pierwszym argumentem funkcji jest identyfikator kolejki. mgsp wskazuje na adres bufora, do którego ma być przekopiowany odbierany komunikat. mgsz 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 od wartości absolutnej mtype
* jeśli msgflg jest ustawiona na **MSG\_EXCEPT**, to otrzymamy dowolny komunikat o typie rożnym od podanego

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\_STAT))

EFAULT (zly adres buf)

EIDRM (kolejka zostala w miedzyczasie skasowana)

EINVAL (zly identyfikator kolejki lub msgsz < 0)

EPERM (brak praw zapisu (IPC\_SET lub IPC\_RMID))

Dopuszczalne komendy to:

* **IPC\_STAT**: uzyskanie struktury msgid\_ds odpowiadającej kolejce (zostaje ona skopiowana pod adres wskazywany przez buf)
* **IPC\_SET**: modyfikacja wartości struktury ipc\_perm odpowiadającej kolejce
* **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\_perm w rozdziale

o cechach wspolnych mechanizmow IPC \*/

msg\_seq+=1; /\* zwiekszenie wartosci globalnej zmiennej zwiazanej z

ipc\_perm.seq - patrz tenze rozdzial \*/

uaktualnienie statystyk;

msgque[msqid]=IPC\_UNUSED;

obudzenie czekajacych na pisanie lub czytanie do/z usuwanej kolejki;

zwolnienie struktur przydzielonych kolejce;

}

POSIX

**POSIX** (ang. ***P****ortable****O****perating****S****ystem****I****nterface for UNI****X***) - 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 mq\_attr \*attr]);

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, którą można zsumować logicznie z wartościami: O\_CREAT, O\_EXCL oraz O\_NONBLOCK (aby używać tych stałych należy dołączyć plik nagłówkowy fcntl.h).

Parametr mode specyfikujemy, gdy tworzymy nową kolejkę i określa on prawa dostępu do niej. Możemy podać wartość ósemkowo lub dowolną sumę logiczną stałych: S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, S\_IWOTH (aby korzystać z tych stałych należy dołączyć plik nagłówkowy sys/stat.h).

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 deskryptorze 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 deskryptorze 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 mq\_attr \*oattr);

Ustawia parametry kolejki o deskryptorze 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 int prio);

Wysyła komunikat wskazywany przez ptr do kolejki o deskryptorze 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 int \*priop);

Odbiera komunikat z kolejki o deskryptorze 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 notifiation jest NULL) mechanizmu powiadomień dla kolejki o deskryptorze 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, SIGEV\_SIGNAL, SIGEV\_THREAD \*/

int sigev\_signo; /\* numer sygnału (dla SIGEV\_SIGNAL) \*/

union sigval sigev\_value; /\* przekazywane procedurze obsługi sygnału lub wątkowi \*/

/\* 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.