Obsługa kolejek komunikatów

Autor: Iwona Madura

19 grudnia 2000

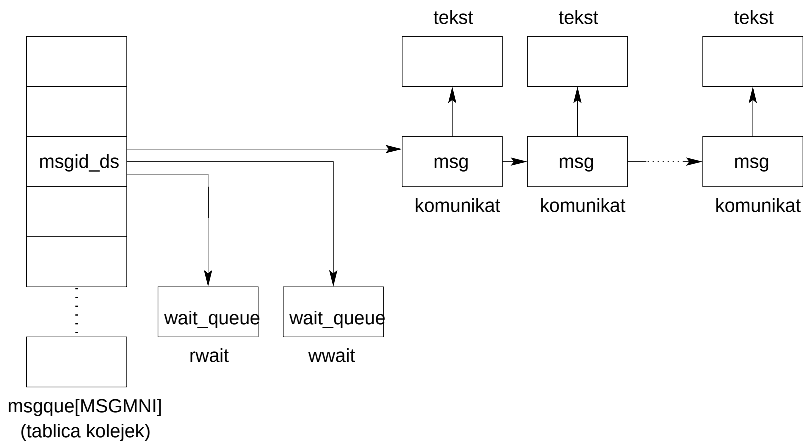
1 Wprowadzenie

Kolejki komunikatów są jednym z trzech mechanizmów IPC. Kolejki komunikatów umożliwiają przesyłanie bloków danych z jednego procesu do drugiego. Jest to dość efektywna i łatwa metoda przesyłania danych między niespokrewnionymi procesami, ograniczona jednak wielkością zarówno pojedynczego komunikatu jak i wszystkich bloków danych w kolejkach systemowych.

2 Struktury danych

2.1 Opis ogólny

Dla każego mechamizmu IPC, w szczególności dla kolejek komunikatów istnieje w przestrzeni jądra jedna globalna tablica (*msgque[MSGMNI]*), której elementami są wskaźniki do pojedynczych struktur. W przypadku kolejek te struktury to *msgid_ds*. *Msgid_ds* z kolei zawiera komunikaty o opisanej niżej strukturze *msg*. Każda kolejka ma swój unikalny klucz oraz desktyptor. Klucze służą do identyfikacji kolejek podczas przesyłania, deskryptor natomiast jest przydzielany przez funkcję *msgget* w celu zmniejszenia prawdopodobieństa odczytu komunikatu przez niepożądany proces (klucze mogą być dowolne w szczególności np. małe liczby całkowite).



2.2 Struktura kolejki komunikatów (msgid_ds)

Dla każdej kolejki w systemie tworzona jest jedna struktura *msgid_ds*. Struktura *msgid_ds* zawiera następujące pola:

ipc_perm informację o prawach dostępu do kolejki oraz jej założycielu

*msg_first wskaźnik do pierwszego komunikatu w kolejce

*msg_last wskaźnik do ostatniego komunikatu w kolejce

msg_stime czas wykonania ostatniej operacji msgsnd

msg_rtime czas wykonania ostatniej operacji msgrcv

msg_ctime czas wykonania ostatniej zmiany

*wwait przydzielona kolejka typu *wait_ queue* na której czekają uśpione procesy zawieszone przy wykonywaniu operacji pisania w danej kolejce komunikatów

*rwait przydzielona kolejka typu *wait_ queue* na której czekają uśpione procesy zawieszone przy wykonywaniu operacji czytania w danej kolejce komunikatów

msg_cbytes bieżącą liczbę bajtów które znajdują się w kolejce

msg_qnum liczbę komunikatów w kolejce

msg_qbytes maksymalną liczbę bajtów w kolejce

msg_lspid pid ostatniego msgsnd

msg_lrpid pid ostatniego msgrcv

2.3 Struktura pojedynczego komunikatu (msg)

Każdemu komunikatowi przyporządkowana jest jedna struktura *msg* zawierająca następujące pola:

*msgnext kolejny komunikat w kolejce

msg_type typ komunikatu (proces nadawczy może nadać wysyłanemu komunikatowi typ (dodatnią liczbę naturalną), a proces odbiorczy może zażądać komunikatu danego typu.

*msg_spot wskaźnik do miejsca w pamięci w którym przechowywana jest właściwa treść komunikatu.

msg_time czas wysyłania danego komunikatu

msg_ts długość treści komunikatu

2.4 Struktura zawierająca wysyłany/otrzymywany komunikat

Komunikat może być przesyłany/otrzymywany przez dowloną strukturę zdefiniowaną przez programistę, pod warunkiem, że jej pierwszym polem będzie wartość typu *long* zawierająca identyfikator typu komunikatu, zaś rozmiar nie przekroczy MSGMAX. Oto przykładowa struktura:

mtype typ komunikatu

mtext[] treść komunikatu

2.5 Ograniczenia

Na dane przesyłane w kolejkach nałożone są limity. W systemie Linux (w innych systemach limity mogą być inne) limity te (określone przez pewne stałe) mają następujące wartości :

- MSGMNB(16384) maksymalny rozmiar całej kolejki
- MSGMAX(4096) maksymalny rozmiar pojedynczego komunikatu
- MSGMNI(128) maksymalna ilość kolejek komunikatów

3 Funkcje obsługujące kolejki komunikatów

3.1 Opis ogólny

Do obsługi (dostępu do) kolejek komunikatów z poziomu aplikacji istnieją cztery funkcje biblioteczne:

- *msgget*(key_t key, int msgflg) - tworzenie kolejki lub dostęp do istniejącej (zwracają desktyptor kolejki używany przez pozostałe funkcje)
- *msgsnd*(int msgid, struct msgbuf *msgp, int msgsz) - wysyłanie komunikatu
- *msgrcv*(int msgid, astruct msgbuf *msgp, int msgsz, long type, int msgflg) - odbieranie komunikatu
- *msgctl*(int msgid, int cmd, struct msgid_ds *buf) - ustawianie i pobieranie parametrów, usuwanie kolejek

Funkcje te w przypadku wersji Linuxa dla PC - i386 wołają funkcję systemową *sys_ipc* przekazując (przez pierwszy rejestr) jako pierwszy parametr identyfikator operacji (MSGSND 11, MSGGET 2, MSGCTL 14, MSGRCV 12 - patrz plik ipc.h) zsumowany logicznie z przesuniętym o 16 bitów w lewo numerem wersji API. W kolejnych parametrach przekazywane są pozostałe argumenty wywołania.

Funkcja *sys_ipc* na podstawie identyfikatora (otrzymanego po obcięciu bitów określających wersję API użytą przez program wołający) wywołuje daną funkcję *sys_msgXXX*.[1] W przypadku innych platform niż PC wołane są bezpośrednio funkcje *sys_msgxxx*.

Ogólna postać funkcji *sys_msgXXX*:

- wchodzą do sekcji krytycznej jądra - *lock_kernel*()(jest to istotna zmiana w stosunku do poprzednich wersji 2.0X przede wszystkim w przypadku systemów wieloprocesorowych)
- wykonują właściwe operacje.
- wychodzą z sekcji krytycznej (*unlock_kernel*)

3.2 Funkcja sys_msgget()

Służy do tworzenia kolejki oraz do uzyskiwania dostępu do istniejącej. Zwraca identyfikator kolejki w przypadku sukcesu bądź jeden z błędów.

Algorytm: int sys_msgget(key_t key, int msgflg)

---

1. *lock_kernel()* - wejście do sekcji krytycznej jądra
2. jeśli była ustawiona flaga IPC_PRIVATE zawsze utwórz kolejkę dostępną tylko dla bieżącego procesu (patrz algorytm *newque*)
3. wpp. sprawdź czy istnieje kolejka o podanym kluczu (*findkey(key)*) Jeśli nie:
   (a) jeśli nie istnieje i została ustawiona flaga IPC_CREAT to utwórz kolejkę (patrz algorytm *newque*)
   (b) jeśli nie istnieje ale flaga IPC_CREAT nie została użyta - zwróć ENOENT (kolejka nie istnieje)
4. jeśli kolejka o danym kluczu już istnieje
   (a) oraz flaga IPC_EXCL została użyta z IPC_CREAT to zwróć EEXISTS (kolejka już istnieje)
   (b) jeśli powyższe flagi nie zostały użyte:
5. otwórz istniejącą już kolejkę *(msgque[id]*)
   (a) sprawdź czy nie została w międzyczasie skasowana (jeśli tak zwróć EIDRM)
   (b) sprawdź czy są do niej odpowiednie prawa (jeśli nie zwróć EACCESS)
6. jeśli sprawdzenia przebiegły pomyślnie:
   (a) zwiększ licznik kolejek w globalnej tablicy msgque.

[1] W przypadku wersji 0 przy msgrcv wykonywane są dodatkowe operacje polegające na wyłuskaniu adresu w pamięci na który ma nastąpić pobranie komunikatu i mające na celu uzgodnienie sposobu wołania przez program obsługi funkcji msgrcv. Dla nowych wersji formaty danych są zgodne.

7. *unlock_kernel()* - wyjście z sekcji krytycznej jądra

Algorytm: int newque(key_t key, int msgflg)

---

1. funkcja znajduje w tablicy globalnej msgque wolną pozycję
2. następnie allokuje wolną strukturę *msqid_ds*
3. odpowiednio ustawia pola *msqid_ds*
4. budzi procesy czekające w *msg_lock*
5. zwiększa licznik w globalnej tablicy *msgque*

3.3 Funkcja sys_msgrcv( )

Służy do pobierania komunikatów z kolejki. Po wejściu do sekcji krytycznej funkcja woła funkcję *reeal_msgrcv* Pierwszym parametrem jest id kolejki, drugim wskaźnik do bufora, trzecim rozmiar, czwartym typ a piątym *msgflg*.

Algorytm: sys_msgrcv(int msgid, struct msgbuf *msgp, size_t msgsz, long msgtyp, int msgflg)

---

1. *lock_kernel();*
2. *real_msgrcv(msgid, msgp, msgsz, msgtyp, msgflg);*
3. *unlock_kernel();*

Algorytm: real_msgrcv(int msgid, struct msgbuf *msgp, size_t msgsz, long msgtyp,int msgflg)

---

1. sprawdź parametry:
   (a) jeśli $msgtyp = 0$ - otrzymujemy najstarszy komunikat
   (b) jeśli $msgtyp > 0$ - otrzymujemy komunikat dango typu
   (c) jeśli $msgtyp < 0$ - otrzymujemy komunikat najmniejszego typu, mniejszego od wartości absolutnej mtype
   (d) jeśli *msgflg* = MSG_EXCEPT - otrzymamy dowolny komunikat o typie różnym od podanego
   (e) jeśli *msgflg* = IPC_NOWAIT w przypadku braku odpowiedniego komunikatu powoduje wyjście z błedęm
   (f) jeśli *msgflg* = MSG_NOWERROR nie powoduje błedu w przypadku gdy komunikat jest za duży (zostaje przekopiowane tyle ile się mieści)
2. znajdź właściwy komunikat
3. sprawdź błędy

(a) E2BIG długość komunikatu większa od *msgsz*

(b) EACCESS brak praw

(c) EFAULT zły adres *msgp*

(d) EINVAL zły id kolejki

4. Zwolnij miejsce w kolejce
5. Obudź procesy czekające w *wwait*
6. Kopiuj komunikat do pamięci procesu przekazanej w parametrze (bufor) i zwolnij zajętą przez niego pamieć w jądrze.
7. Jeśli komunikatu nie znaleziono (ENOMSG)i była ustawiona flaga NO_WAIT, jeśli pojawił się sygnał to zwracane jest RINTERRUPT, jeśli nie to proces zasypia.
8. Po obudzeniu wykonuje wszystko od nowa (poza sprawdzeniem parametrów)

3.4 Funkcja sys_msgsnd()

Służy do wysłania komunikatu do kolejki. Komunikat musi być mniejszy od systemowego maksimum oraz zaczynać się od wartości *long int*, która będzie użyta jako id typu komunikatu w funkcji odbiorczej. Po wejściu do sekcji krytycznej funkcja woła funkcję *real_msg*. Pierwszy parametr to identyfikator kolejki, drugi (*msgptr*) wskaźnik do wysyłanego komunikau, trzeci *msgsz* jest rozmiarem komunikatu (nie licząc typu komunikatu), a czwarty to *msgflg*.

Algorytm: sys_msgsnd(int msgid, struct msgbuf *msgp, size_t msgsz, int msgflg)

---

1. *lock_kernel();*
2. *real_msgsnd(msgid, msgp, msgsz, msgflg);*
3. *unlock_kernel();*

Algorytm: real_msgsnd(int msgid, struct msgbuf *msgp, size_t msgsz, int msgflg)

---

1. Sprawdź:
2. poprawność wysyłanego komunikatu: jeśli wysyłany komunikat ma nieprawidłowy identyfikator (*msgid* < 0) rozmiar (np $>$ MSGMAX lub < 0) lub typ - zwróć EINVAL, a jeśli na podany komunikat brakuje pamięci to zwróć ENOMEM.

   (a) prawa dostępu: jeśli prawa dostępu są nieodpowiednie zwróć EACCESS

   (b) ilość miejsca w kolejce: jeśli kolejka jest pełna zwróć EAGAIN

3. jeśli miejsce w kolejce i gdy sprawdzenia przebiegły pomyślnie w kolejce zajmowane jest miejsce na komunikat i komunikat jest przepisywany z pamięci procesu wołającego do wewnętrznej pamięci jądra. Następnie budzone są procesy czekające w kolejce rwait.

4. jeśli nie ma miejsca w kolejce (i nie jest ustawione IPC_NOWAIT) uśpij proces na kolejce *wwait*

5. proces może zostać obudzony sygnałem i wówczas jest przerywany z wartością EINTR) lub przez pojawienie się miejsca w kolejce (wakeup)

6. po obudzeniu jeszcze raz sprawdzane są prawa dostępu i ilość miejca (jeśli brakuje to śpimy znowu)

7. w przypadku gdy sprawdzenia przebiegły pomyślnie w kolejce zajmowane jest miejsce na komunikat i komunikat jest przepisywany z pamięci procesu wołającego do wewnętrznej pamięci jądra. Następnie budzone są procesy czekające w kolejce *rwait*.

3.5 Funkcja sys_msgctl()

Funkcja służy do odczytu i modyfikacji różnych właściwości kolejki.

Algorytm: sys_msgctl(int msgid, int cmd,struct msqid_ds *buf)

---

1. *lock_kernel();*

2. Sprawdzenie parametrów. Gdy cmd może przyjmuje następujące wartości wykonywane są następujące czynności:

   (a) MSG_STAT - wyświetl statystyki

   (b) IPC_SET - modyfikacja wartości struktury ipc_perm odpowiadającej kolejce

   (c) IPC_STAT uzyskanie struktury *msgid_ds* odpowiadającej kolejce (zostaje ona skopiowana pod adres wskazany przez *buf*)

   (d) IPC_RMID - po sprawdzeniu uprawnień funkcja *freeque* zaznacza kolejkę jako nieużytą, zmniejsza licznik użytych kolejek . jesli jakieś procesy czekały na odczyt albo zapis to są budzone. Po obudzeniu procesów wywołany jest scheduler kernela, potem z kolejki są usuwane komuniakty i zwalniana ich pamięć - na końcu zwalniana jest pamięć kolejki.

   (e) IPC_INFO, MSG_INFO

3. Sprawdzenie błędów:

   (a) EACCES - brak praw odczytu (IPC_STAT)

   (b) EFAULT - zły adres bufora

   (c) EIDRM - kolejka została w międzyczasie skasowana

   (d) EINVAL - zły id kolejki lub *msgsz* < 0

(e) EPERM - brak praw zapisu (IPC_SET lub IPC_RMID)

4. *unlock_kernel();*

---

4 Bibliografia

1. Pliki źródłowe

   (a) include/linux/msg.h

   (b) linux/ipc/msg.c

   (c) linux/arch/i386/kernel/sys_i386.c

2. Neli Matthew "Linux programowanie".

3. Maurice J.Bach Budowa systemu operacyjnego UNIX

Spis treści