Segmenty pamięci dzielonej IPC

Stanisław Miszczak

30 grudnia 2000 r.

1 Wprowadzenie

Systemy operacyjne z rodziny unix (w tym także i **Linux**) niemal od początku swego istnienia umożliwiają i wspierają jednoczesne wykonywanie kilku programów. Nie jest to możliwe bez zaawansowanych mechanizmów ochrony i wykluczania dostępu do zasobów komputera. Ochrona polega między innymi na izolowaniu programów od siebie: są one umieszczane w osobnych (wirtualnych) przestrzeniach adresowych procesora i nie mogą bezpośrednio komunikować się z pozostałymi. Całkowita izolacja uniemożliwiłaby jednak poprawne funkcjonowanie komputera - programy muszą mieć możliwość wymieniania informacji ze sobą. Wszystkie aplikacje typu klient-serwer (np. poczta elektroniczna, www) składają się z dwóch (lub większej ilości) programów, pomiędzy którymi przesyłane są dane. Wymiana informacji zachodzi też pomiędzy wieloma elementami systemu operacyjnego (np. dowolny program i syslogd - proces zbierający komunikaty informacyjno-diagnostyczne i zapisujący je do pliku). Dlatego właśnie zaistniała potrzeba stworzenia szybkich i wygodnych mechanizmów komunikacji międzyprocesowych. W systemie Linux istnieje wiele takich mechanizmów - na przykład **IPC**.

1.1 Mechanizmy IPC

IPC (ang. *Inter-Process Comunication*) jest mechanizmem komunikacji, po raz pierwszy zastosowanym w systemie V, a dziś praktycznie dostępnym w każdym systemie z rodziny unix (także w Linuksie). Na IPC składają się trzy, właściwie niezależne, sposoby komunikacji: kolejki komunikatów, semafory i segmenty pamięci dzielonej. Wszystkie te mechanizmy mają podobną filozofię i w podobny sposób się z nich korzysta. Zasoby IPC są dostępne globalnie dla całego systemu - identyfikuje się je za pomocą unikalnych numerów - kluczy. Zaletą takiego rozwiązania jest to, że komunikacja nie jest ograniczona do dwóch programów, wadą zaś - słaba ochrona przed niepowołanym użyciem. Dostęp do zasobów odbywa się poprzez specjalne funkcje systemowe. Aby nawiązać komunikację, jeden z programów musi wpierw utworzyć zasób o wybranym, ustalonym kluczu. Drugi program żąda uzyskania dostępu, podając ten sam klucz. Po pomyślnym przebiegu powyższych operacji, oba programy uzyskują id nowoutworzonego zasobu. Dzięki temu mają możliwość wymiany informacji - w tym celu wołają odpowiednie funkcje systemowe, z parametrem id zasobu. Po zakończeniu komunikacji, program, który utworzył zasób, musi go jawnie usunąć - nie dzieje się to automatycznie

! Listę aktualnie wykorzystywanych zasobów IPC w systemie można uzyskać za pomocą polecenia ipcs. Administrator ma możliwość usuwania niepotrzebnych zasobów.

2 Segmenty pamięci dzielonej

Pamięć dzielona, jak sama nazwa wskazuje, polega na uwspólnieniu bloku pamięci operacyjnej pomiędzy dwoma, lub większą ilością procesów. Po utworzeniu i uzyskaniu dostępu do segmentu pamięci dzielonej, program może ją włączyć do swojej przestrzeni adresowej i korzystać z niego jak z każdego innego bloku pamięci. Program może więcej niż jeden raz dołączyć dany blok. Należy jednak pamiętać, że segment dzielony sam z siebie nie zapewnia wykluczania dostępu. Musi o to się zatroszczyć programista, np. realizując *algorytm czytelników i pisarzy* za pomocą semaforów IPC. Proces potomny dziedziczy wszystkie dołączenia segmentów procesu macierzystego (po wykonaniu funkcji `fork()`). Wywołanie funkcji `exit()` lub `exec()` powoduje odłączenie wszystkich dołączonych segmentów, ale nie kasuje ich fizycznie. Usunięcie segmentu następuje po jawnym wywołaniu odpowiedniej funkcji - jeśli jednak licznik dołączeń jest niezerowy, segment jest oznaczany jako "gotowy do usunięcia", a fizyczne skasowanie nastąpi, gdy ostatni program odłączy go ze swojej przestrzeni adresowej.

3 Interfejs programisty

Do korzystania z segmentów pamięci dzielonej IPC służą następujące funkcje systemowe:

- `int shmget(key t_key, int size, int shmflg);`
- `int shmctl(int shmid, int cmd, struct shmid_ds *buf);`
- `void * shmat(int shmid, const void * shmaddr, int shmflg);`
- `int shmdt(const void * shmaddr);`

Struktura `shmid_ds` używana przez funkcję `shmctl()` ma następującą postać:

```
struct shmid_ds {
    struct ipc_perm shm_perm;   /* prawa dostępu */
    int shm_segsz;              /* rozmiar w bajtach */
    time_t shm_atime;           /* ostatni czas dołączenia */
    time_t shm_dtime;           /* ostatni czas odłączenia */
    time_t shm_ctime;           /* ostatni czas operacji ctl */
    unsigned short shm_cpid;    /* pid procesu tworzącego */
    unsigned short shm_lpid;    /* pid ostatnio odw. się procesu */
    short shm_nattch;           /* liczba dołączeń */
};
```

3.1 `shmget()`

Funkcja ta służy do tworzenia nowego lub uzyskiwania dostępu do istniejącego segmentu pamięci dzielonej. Pierwszym argumentem (`key`) jest klucz, czyli ustalona wartość, znana komunikującym się programom, służąca do identyfikacji segmentu. W przypadku, gdy segment chcemy utworzyć, trzeba podać jego rozmiar jako drugi parametr funkcji (`size`), zaś wśród flag (`shmflg`) należy wyspecyfikować `IPC_CREAT`. Jeśli dodatkowo użyjemy flagi `IPC_EXCL`, funkcja zwróci błąd, w przypadku, gdy segment już istniał. Po pomyślnym utworzeniu segmentu, zwracany jest jego id, wykorzystywany przez dalsze funkcje. Przy tworzeniu można też użyć specjalnej wartości klucza `IPC_PRIVATE`, która w połączeniu z `IPC_CREAT` spowoduje stworzenie nowego segmentu (Jednakże w takiej sytuacji trzeba w jakiś sposób przekazać id zasobu pozostałym komunikującym się programom) Flagi mogą też zawierać prawa dostępu do segmentu, przy czym istotne są tylko prawa zapisu i odczytu.

3.2 `shmctl()`

Funkcja realizuje kilka operacji, jakie można wykonać na istniejącym segmencie pamięci dzielonej. Pierwszym argumentem jest id segmentu, drugim (`cmd`) - polecenie, jaką chcemy wykonać. Jeśli wartością polecenia jest `IPC_STAT`, wówczas informacje o segmencie zostaną skopiowane do miejsca w pamięci wskazywanego przez trzeci argument funkcji `buf` (patrz format struktury shmid_ds). Analogicznie, polecenie `IPC_SET` spowoduje zaktualizowanie niektórych pól struktury opisującej segment na podstawie danych wskazywanych przez `buf`. Na koniec, polecenie `IPC_RMID` spowoduje zaznaczenie segmentu jako gotowego do usunięcia (i usunie go, jeśli licznik dowiązań jest równy zero).

3.3 `shmat()`

Funkcja ta służy do włączenia segmentu pamięci dzielonej do przestrzeni adresowej programu. Oprócz id segmentu, kluczowym parametrem funkcji jest pole `shmaddr`. Jeśli wynosi ono zero, system sam wyszuka wolnego niezamapowanego miejsca pod które może włączyć segment. W przeciwnym razie (i jeśli wyspecyfikowano flagę `SHM_RND`) próbuje włączyć segment w miejsce wskazywane przez `shmaddr` (z dokładnością do wielokrotności stałej `SHMLBA`). Wartością zwracaną przez funkcję jest adres segmentu, bądź -1, jeśli wystąpił błąd. Należy zwrócić uwagę, że można dołączać segmenty, które wcześniej były zaznaczone jako gotowe do usunięcia.

3.4 `shmdt()`

Funkcja odłącza segment pamięci dzielonej z przestrzeni adresowej programu. Jedynym argumentem jest adres, który zwróciła funkcja `smmat()`. Należy zwrócić uwagę, że jeśli segment był zaznaczony jako gotowy do usunięcia, a dane dołączenie było ostatnim, wówczas wywołanie funkcji `shmdt()` spowoduje usunięcie segmentu z systemu.

4 Implementacja w jądrze systemu

Implementacja segmentów pamięci dzielonej IPC nie uległa znaczącym zmianom od roku 1990, jednakże poniższy opis opiera się na wersji 2.2.17 jądra systemu Linux.Podstawową strukturą wewnętrzną jądra opisującą pojedynczy segment pamięci dzielonej IPC jest `shmid_kernel`:

```
struct shmid_kernel {
    struct shmid_ds u;
    unsigned long shm_npages;     /* rozmiar segmentu (w stronach) */
    unsigned long *shm_pages;     /* tablica wskaźników do ramek stron */
    struct vm_area_struct *attaches;      /* dekskryptory dołączeń */
};
```

Obiektem grupującym informacje o wszystkich segmentach pamięci dzielonej IPC dostępnych w systemie jest `struct shmid_kernel *shm_segs[SHMMNI]`, przy czym stała `SHMMNI` to maksymalna ilość segmentów, jakie można utworzyć. Wartościami w tablicy `shm_segs` są wskaźniki do struktur lub dwie wartości specjalne: `IPC_UNUSED` i `IPC_NOID`. Pierwsza z nich oznacza, że dane pole tablicy nie opisuje żadnego istniejącego segmentu. Druga (`IPC_NOID`) jest używana wewnętrznie przy tworzeniu nowego segmentu (patrz poniższy ustęp)

4.1 Tworzenie segmentu

Funkcja systemowa `shmget()` korzysta wewnętrznie z dwóch pomocniczych funkcji: `newseg()` oraz `findkey()`. Pierwsza z nich znajduje wolne pole tablicy `shm_segs` (równe `IPC_UNUSED`), zmienia jego wartość na `IPC_NOID` i rozpoczyna proces tworzenia segmentu. Proces ten polega na przydzieleniu pamięci na strukturę `shmid_kernel` (funkcja `kmalloc()`) oraz przydzieleniu stron składających się na przyszły segment pamięci dzielonej (funkcja `vmalloc()`). Po wypełnieniu struktury, wskaźnik do niej zastępuje wartość `IPC_NOID` w polu tablicy `shm_segs`. Budzony jest także ewentualny proces zawieszony na kolejce typu `wait_queue` (patrz dalszy opis). Funkcja `findkey()` wywoływana jest, gdy wartość klucza była różna od `IPC_PRIVATE`. W takiej sytuacji tablica `shm_segs` skanowana jest liniowo od początku w poszukiwaniu opisu segmentu o kluczu pasującym do danego. W przypadku, gdy wartość pola tablicy wynosi `IPC_NOID`, funkcja zawiesza się na kolejce typu `wait_queue`. Gdy segment pamięci o pasującym kluczu został znaleziony, zwracany jest jego id.

4.2 Modyfikowanie segmentu

Polecenie `IPC_STAT` funkcji `shmctl()` polega na skopiowaniu pola `shmid_ds d` struktury `shmid_kernel` opisującej dany segment do przestrzeni adresowej użytkownika (`copy_to_user()`). Z kolei, polecenie `IPC_SET` kopiuje jedynie pole `shm_perm` do struktur systemowych.
Okazuje się, że funkcja `shmctl()` ma dwa nieudokumentowane w manualach polecenia: `IPC_LOCK` oraz `IPC_UNLOCK`. Służą one do zablokowania i odblokowania segmentu w pamięci - uniemożliwiają wyswapowanie go na dysk. W tym celu ustawiana jest specjalna flaga `SHM_LOCKED` w polu `mode` struktury `shm_perms` opisującej prawa dostępu do segmentu.

4.3 Mapowanie segmentu

Funkcje `shmat()` oraz `shmdt()` są najbardziej złożone pod względem wielkości użytego kodu, gdyż konieczne jest pracochłonne przygotowanie systemowych struktur, jednakże zasada ich działania jest zupełnie prosta. Funkcja `shmat()` musi wpierw znaleźć wolny obszar w przestrzeni adresowej programu (`get_unmapped_area()`) lub sprawdzić, czy adres podany jako parametr jest możliwy do wykorzystania (`find_vma_intersection()`). Następnie musi przygotować systemową strukturę `vm_operations_struct shm_vm_ops` i wykonać funkcje `insert_vm_struct()` i `merge_segments()`. Na koniec należy zaktualizować pola struktury `shmid_kernel` w taki sposób, żeby pole `attaches` zawierało bieżące dołączenie segentu. Dokładny opis działania wymienionych funkcji operujących na pamięci wirtualnej (mm) nie jest przedmiotem niniejszego opracowania - polecam lekturę kodu znajdującego się w katalogu /usr/src/linux/mm.

4.4 Usuwanie segmentu

Usunięcie segmentu może nastąpić w dwóch przypadkach: w poleceniu `IPC_RMID` funkcji `shmctl()` lub po wywołaniu `shmdt()`, gdy segment był zaznaczony jako gotowy do usunięcia. Owo zaznaczenie realizuje się poprzez ustawienie flagi `SHM_DEST` w polu `mode` struktury `shm_perms` opisującej prawa dostępu do segmentu. Fizycznym usunięciem zajmuje się funkcja `killseg()`, która zwalnia wszystkie strony składające się na segment, także te, które w danym momencie zostały wyswapowane na dysk (funkcje `free_page()` i `swap_free()`). Niszczona jest także sama struktura `shmid_kernel` (`kfree()`). Po usunięciu wszystkich informacji związanych z segmentem, odpowiednie pole tablicy `shm_segs` przyjmuje wartość `IPC_UNUSED`.

5 Uwagi końcowe

Podstawową zaletą pamięci dzielonej jest jej szybkość: kopiowanie danych z i do pamięci odbywa się wielokrotnie szybciej niż np. z pamięci na dysk. Wadą jest konieczność stosowania dodatkowych mechanizmów ochrony w celu zapewnienia wykluczania. Można to robić za pomocą semaforów IPC, co zresztą jest zalecane, ale powoduje, że narzut związany z wywoływaniem funkcji systemowych staje się znaczący. Dlatego też korzystanie z pamięci dzielonej IPC ma sens w przypadku przesyłania bardzo dużych ilości danych (rzędu setek kilobajtów, albo nawet megabajtów).

Warto też zwrócić uwagę na możliwość skasowania segmentu pamięci dzielonej tuż po pierwszym dołączeniu do jakiegoś programu. Dzięki temu nie trzeba się martwić usuwaniem segmentu w przypadku, gdyby proces niespodziewanie zginął.

Ostatnia uwaga dotyczy implementacji: w typowym systemie Linuksowym wartość `SHMMNI` wynosi 4096, co oznacza, że tablica `shm_segs` zajmuje około 16 KB pamięci, nawet jeśli w systemie nie ma żadnych segmentów. Teoretycznie, można by tą tablicę zastąpić listą, jednakże wydłużyłoby to czas dostępu do niej, a co za tym idzie, czas działania wszystkich funkcji operujących na segmentach.