Obsługa semaforów (IPC)

Szymon Janus

20 grudnia 2000

1 Wprowadzenie

Semafory są mechanizmem umożliwiającym synchronizację procesów, ochronę współdzielonych zasobów. Można je zdefiniować jako zmienne całkowitoliczbowe przyjmujące wartości nieujemne, na których zdefiniowane są operacje: inicjacji, opuszczenia (ozn. P(s)) i podniesienia (ozn. V(s)). Operacje te mają następującą semantykę:

init(S,n): $S := n$

P(S): gdy $S > 0$, wtedy $S := S - 1$,
wpp proces jest zawieszany w oczekiwaniu na ten warunek,

V(S): $S := S + 1$.

Zakłada się, że implementacja powyższych operacji zapewnia żywotność.

Realizacja semaforów w systemie Linux dość znacznie odbiega od powyższej definicji. Podstawowe różnice to:

- możliwość atomowego operowania na zestawach semaforów,
- możliwość podnoszenia i opuszczania semafora o dowolną wartość,
- możliwość sprawdzania i zmiany wartości pojedynczego semafora,
- możliwość zawieszania procesów w oczekiwaniu na wyzerowanie semafora,
- możliwość sprawdzenia ilości procesów oczekujących na określony stan semafora,
- możliwość wykonania operacji nieblokującej,
- możliwość cofnięcia w momencie zakończenia procesu wskazanej części lub wszystkich operacji dokonanych przez ten proces,
- brak zapewnienia żywotności.

Dla semaforów, tak jak dla innych mechanizmów IPC (InterProcess Communication) dostarczanych przez jądro, można określić prawa dostępu. Podobnie jak w przypadku plików oddzielnie określane są prawa dla właściciela, grupy i innych użytkowników. Prawa te, to prawo zapisu i prawo odczytu. Określane dla plików prawo do wykonania nie jest w przypadku semaforów wykorzystywane. Prawo zapisu (zmiany) umożliwia praktyczne korzystanie z semafora, czyli podnoszenie i opuszczanie, a także część operacji kontrolnych. Prawo odczytu pozwala na zawieszanie procesu w oczekiwaniu na wyzerowanie semafora oraz na część operacji kontrolnych.

Poniższe opracownie opisuje implementację zawartą w wersji jądra o numerze 2.2.17.

2 Struktury danych

Definicje poniższych struktur można znaleźć w pliku `include/linux/sem.h`.

2.1 Struktura `semid_ds`

Struktura `semid_ds` opisuje jeden zestaw semaforów.

`struct ipc_perm sem_perm;`	— prawa dostępu (`include/linux/ipc.h`),
`__kernel_time_t sem_otime;`	— czas ostatniej operacji `sys_semop`,
`__kernel_time_t sem_ctime;`	— czas ostatniej modyfikacji,
`struct sem *sem_base;`	— wskaźnik do tablicy semaforów,
`struct sem_queue *sem_pending;`	— operacje do wykonania,
`struct sem_queue **sem_pending_last;`	— ostatnia z operacji do wykonania,
`struct sem_undo *undo;`	— lista operacji cofających,
`unsigned short sem_nsems;`	— ilość semaforów w zestawie.

2.2 Struktura `sem`

Pojedynczy semafor jest opisany przez strukturę `sem`.

`int semval;` — wartość,

`int sempid;` — `pid` procesu, który dokonał ostatniej operacji.

2.3 Struktura `sem_queue`

Struktura `sem_queue` opisuje element kolejki atomowych operacji oczekujących na wykonanie.

`struct sem_queue *next;` — następny element kolejki,
`struct sem_queue **prev;` — wskaźnik na wskaźnik na ten element w poprzedniku,
`struct wait_queue *sleeper;` — śpiący proces,
`struct sem_undo *undo;` — struktura cofnięć,
`int pid;` — pid śpiącego procesu,
`int status;` — stan zakończenia operacji,
`struct semid_ds *sma;` — tablica semaforów, której dotyczy kolejka,
`struct sembuf *sops;` — tablica operacji do wykonania,
`int nsops;` — ilość operacji,
`int alter;` — czy operacje modyfikują semafory?

2.4 Struktura `sembuf`

Struktura `sembuf` opisuje pojedynczą operację na jednym semaforze.
`unsigned short sem_num;` — numer semafora w zestawie,
`short sem_op;` — operacja,
`short sem_flg;` — flagi.
Operacja to wielkość, o którą chcemy zmienić wartość semafora, lub zero, gdy oczekujemy na jego wyzerowanie.

2.5 Struktura `sem_undo`

Struktura `sem_undo` opisuje sumaryczny efekt operacji dokonanych przez proces i zaznaczonych jako "do cofnięcia" (flaga `SEM_UNDO`) na wskazanym zestawie semaforów, dla każdego semafora oddzielnie.
`struct sem_undo *proc_next;` — następna struktura dla tego procesu,
`struct sem_undo *id_next;` — następna struktura dla tego zestawu semaforów,
`int semid;` — identyfikator zestawu semaforów,
`short *semadj;` — tablica zmian (dla każdego z semaforów).

3 Zmienne globalne

Stała `SEMMNI` wykorzystywana poniżej określa maksymalną ilość zestawów semforów.
`struct semid_ds *semary[SEMMNI];` — tablica zestawów semaforów,
`int used_sems;` — ilość semaforów,
`int used_semids;` — ilość zestawów semaforów,
`struct wait_queue *sem_lock;` — procesy oczekujące na modyfikację tablicy,
`int max_semid;` — najwyższy wykorzystywany indeks w tablicy,
`int sem_seq;` — zmienna wykorzystywana dla zapewniania unikalności identyfikatorów.

4 Funkcje i ich realizacja

Definicje poniższych funkcji znajdują się w pliku `ipc/sem.c`.

4.1 Funkcja `sys_semget`

Funkcja `sys_semget` służy do stworzenia nowego zestawu semaforów lub pobrania identyfikatora istniejącego zestawu. Funkcja `semget` biblioteki standardowej jest po prostu interfejsem dla tej funkcji.

Algorytm 1 sys_semget

Dane wejściowe: klucz, ilość semaforów, flagi (w tym prawa dostępu)
Wynik: identyfikator zestawu lub błąd

```
lock_kernel(); {por. „Obsługa wieloprocesorowości”}
jeśli trzeba poszukać klucza to
    findkey(klucz);
jeśli trzeba stworzyć zestaw to
    newary(klucz, ilość semaforów, flagi);
unlock_kernel(); {j.w.}
```

Funkcja `sys_semget` woła dwie funkcje `findkey` i `newary`. Pierwsza z nich stara się znaleźć zestaw semaforów o podanym kluczu. Druga natomiast, tworzy, o ile jest to możliwe, nowy zestaw semaforów.

Algorytm 2 findkey

Dane wejściowe: klucz
Wynik: identyfikator lub -1

```
dla i = 0 do max_semid wykonuj
    dopóki semary[i] jest w trakcie modyfikacji wykonuj
        zaśnij w kolejce sem_lock;
    jeśli semary[i] ma szukany klucz to
        zakończ zwracając i;
zwróć −1;
```

Dłuższego komentarza wymaga zwracany przez funkcję `newary` identyfikator. Jest to podany klucz powiększony o iloczyn `SEMMNI` (maksymalna ilość zestawów) i wartości zmiennej globalnej `sem_seq`. Zmienna ta jest zwiększana o 1 przy każdym usuwaniu jakiegokolwiek z zestawów. Tak więc dwa zestawy trafiające pod ten sam indeks tablicy mają na pewno różne identyfikatory. Zapewnia to więc ich unikalność.

4.2 Funkcja `sys_semop`

Funkcja `sys_semop` służy do wykonywania zasadniczych operacji na zestawach semaforów. Przyjmuje ona jako jeden z parametrów listę operacji do wykonania. Pojedyncza operacja to podniesienie lub opuszczenie wskazanego semafora o pewną niezerową wartość (w szczególności większą od 1). Operacją może być także ewentualne zawieszenie się w oczekiwaniu na wyzerowanie semafora.

Pomocnicza funkcja `update_queue` wyszukuje zawieszone operacje, które mogłyby być w danej chwili wykonane i budzi odpowiednie procesy.

Algorytm 3 newary

Dane wejściowe: klucz, ilość semaforów, flagi

Wynik: identyfikator stworzonego zestawu lub błąd

```
jeśli ilość żądanych jest niepoprawna lub zostałby przekroczony limit to
   zakończ z błędem;
dla id = 0 do SEMMNI wykonuj
   jeśli semary[id] jest nieużywane to
      zaznacz, że jest modyfikowane;
      zakończ pętlę; {id zawiera znaleziony indeks}
jeśli nie znaleziono wolnego pola tablicy to
   zakończ z błędem;
zaalokuj pamięć dla struktury sem_ds i tablicy semaforów; {w spójnym kawałku}
jeśli nie udało się zaalokować pamięci to
   zakończ z błędem;
inicjuj pola struktury; {najpierw wypełnij zerami}
ustaw prawa dostępu;
uaktualnij zmienne globalne;
obudź procesy śpiące w kolejce sem_lock;
zwróć identyfikator; {patrz opis}
```

Warto w tym miejscu wspomnieć o braku zapewniania żywotności. Bierze się on z faktu, że funkcja `update_queue` przegląda listę oczekujących operacji od początku i budzi te procesy, których operacje mogą być w danym momencie wykonane. Jeżeli operacje polegają na zmianach semaforów o duże wartości lub na zmianach wielu semaforów może się zdarzyć, że zawsze znajdzie się proces o mniejszych wymaganiach i to on będzie odwieszony.

Jeżeli zestaw składa się z jednego semafora, a wszystkie operacje modyfikują jego wartość o 1 ("semafor klasyczny") mamy do czynienia z kolejką typu FIFO.

Funkcja `semop` biblioteki standardowej jest interfejsem tej funkcji (por. opis funkcji `sys_semget`).

Jeżeli trzeba zasnąć, a funkcja wywołana jest w sposób nieblokujacy, zwracany jest błąd `EAGAIN`.

4.3 Funkcja `sys_semctl`

Funkcja `sys_semctl` służy do wykonywania rozmaitych funkcji kontrolnych. Interfejsem do niej jest funkcja `semctl` biblioteki standardowej. Szczegóły wywołania i semantyki można znaleźć w podręczniku systemu (`man semctl`). Funkcja ta zrealizowana jest w postaci kilku bardzo rozbudowanych instrukcji wyboru nie robiących faktycznie nic ciekawego. Jej dokładny opis nie wydaje się konieczny.

4.4 Funkcja `sys_exit`

Funkcja `sys_exit` wywoływana jest przez funkcję `do_exit`, czyli przy kończeniu procesu. Jej działanie sprowadza się do cofnięcia, na podstawie informacji zgromadzonych w

Algorytm 4 sys_semop

Dane wejściowe: identyfikator, lista operacji, ilość operacji
Wynik: błąd lub 0

```
lock_kernel();
jeśli jakikolwiek z parametrów jest niepoprawny to
   zakończ się z odpowiednim błędem
skopiuj tablicę operacji do własnej zmiennej;
przejrzyj operację i zapamiętaj czy są podniesienia, opuszczenia, występuje flaga SEM_UNDO;
{wiemy już jakich praw potrzebuje użytkownik}
jeśli użytkownik nie ma odpowiedniego prawa to
   zakończ z błędem;
jeśli występuje flaga SEM_UNDO to
   jeśli proces ma listę cofnięć (unieważnień) to
      znajdź strukturę dla naszego zestawu
   jeśli proces nie ma listy cofnięć lub nie ma nas na niej to
      dodaj do listy strukturę dla naszego zestawu
error := try_atomic_semop(...); {spróbuj wykonać zadane operacje}
jeśli error oznacza błąd to
   zakończ z tym błędem;
w przeciwnym przypadku jeśli error == 0 to
   update_queue(...)
   zakończ i zwróć 0;
w przeciwnym przypadku
   zainicjuj zmienną do wstawienia do kolejki;
   jeśli operacje zmieniają wartości to
      dodaj się na koniec kolejki;
   w przeciwnym przypadku
      dodaj się na początek kolejki; {tam lepiej czekać na wyzerowanie}
wykonuj w pętli
   zaśnij;
   jeśli status == 1 to
      error := try_atomic_semop(...);
      jeśli error ≤ 0 to
         wyjdź z pętli;
   w przeciwnym przypadku
      error := status;
      jeśli element nie został usunięty z kolejki to
         usuń element z kolejki;
unlock_kernel();
zwróć error;
```

Algorytm 5 update_queue

Dane wejściowe: zestaw semaforów

```
dla wszystkich elementów kolejki oczekujących operacji wykonuj
  jeśli jest już obudzona to
    następny obrót pętli;
  error:=try_atomic_semop(...); {dla tej operacji}
  jeśli error ≤ 0 to
    obudź proces;
    jeśli error == 0 i operacje modyfikują semafory to
      status := 1; {niech proces będzie samodzielny}
      zakończ procedurę;
    w przeciwnym przypadku
      {operacja wywołałaby błąd lub jest oczekiwaniem na 0}
      status := error;
      usuń element z kolejki;
```

Algorytm 6 try_atomic_semop

Dane wejściowe: zestaw semaforów, tablicę operacji, ilość operacji, strukturę cofnięć, pid, czy cofać?

Wynik: 0 — można, 1 — trzeba zasnąć, reszta — błąd

```
dla wszystkich operacji w tablicy wykonuj
  jeśli operacja jest zerowa i wartość semafora niezerowa to
    zakończ z 1 lub EAGAIN;
  wykonaj operację;
  jeżeli trzeba odnotuj ją w strukturze cofnięć;
  jeśli wartość semafora jest ujemna to
    zakończ z 1 lub EAGAIN;
  jeśli wielkość semafora przekracza dopuszczalny zakres to
    zakończ z błędem;
  jeśli czy cofać? to
    cofnij;
    zakończ z 0;
  jeśli kończysz z 1 lub błędem, a wykonano operacje to
    cofnij;
```

strukturach `sem_undo`, operacji wykonanych przez proces. Cofnięcie polega na dodaniu do semaforów wartości z tablic `semadj` wspomnianych wyżej struktur. Jeżeli dla któregoś z semaforów otrzymana wartość jest ujemna, jest on zerowany. Po każdej takiej operacji wykonywana jest funkcja `update_queue`.

W komentarzach wyrażono wątpliwość czy takie podejście jest właściwe. Czy nie byłoby właściwsze zawieszenie się w oczekiwaniu na sytuację, w której cofnięcie można wykonać "bezboleśnie" tj. bez sztucznego ponoszenia semaforów. Takie podejście jest ponoć realizowane w "oryginalnej implementacji" (System V?).

Takie, a nie inne rozwiązanie, wg komentarzy w pliku `ipc/sem.c`, było początkowo subiektywnym wyborem autora implementacji. Inny z programistów dodatkowo przyznał temu rozwiązaniu zgodność ze standardem POSIX.

5 Ciekawostka

Na 812 linii kodu w pliku `ipc/sem.c` instrukcję `goto` można znaleźć w 49 miejscach.

6 Źródła

- kod źródłowy jądra 2.2.17, pliki `include/linux/sem.h`,`ipc/sem.c`, przeglądane zestawem skryptów ze strony `http://lxr.linux.no`,
- podręcznik systemowy (`man`),
- slajdy do wykładu z Programowania Współbieżnego (definicja semafora),
- opracowania z lat poprzednich znalezione na stronie wykładu.