MECHANIZMY SYNCHRONIZACJI IPC

Omawiane do tej pory rozwiązanie semaforów pochodzi z standardu ***UNIX System V*** i jest zasadniczo ukierunkowane na bardzo poważne zastosowania, kiedy dla realizacji określonego zadania wymagane jest rozstrzygnięcie bardzo złożonych problemów synchronizacji procesów. Chociaż zawiera się on w standardzie ***POSIX***, to jednak ten ostatni oferuje również i całkowicie odmienne podejście do problemu (choć dające docelowo identyczne możliwości zastosowań), równocześnie nie są elementem interface'u programowania ***IPC***.

- ***POSIX*** zakłada w procesie jego utworzenia zawsze pojedynczy semafor, podczas gdy ***SysV*** zbiorowość semaforów;
- ***POSIX*** oferuje możliwość definiowania semaforów jako nazwanych lub nienazwanych pierwsze dają możliwość synchronizacji między całkowicie niepowiązanymi procesami, natomiast drugie umiejscawiane są w obrębie segmentu dzielonego pamięci;
- ***POSIX***'owe przyjmują nieujemne wartości licznika i może być on zmieniany wyłącznie o ± 1, w ***SysV*** nie istnieją tego rodzaju ograniczenia;
- ***POSIX*** nie daje możliwości ingerowania wobec uprawnień odnośnie użycia semafora
- semafor ***POSIX***, w zamyśle twórców, odnosi się raczej do pojedynczego procesu, natomiast semafory ***SysV*** to semafory systemowe;
- semafory ***POSIX*** są znacznie bardziej efektywne, w sensie wydajności, niż ***SysV***;
- semafory ***POSIX*** są jednak - z punktu widzenia użytkownika - znacznie bardziej elementarne i poręczne.

MECHANIZMY SYNCHRONIZACJI IPC

Chęć użycia semafora nazwanego wiąże się z uprzednim jego utwarciem **sem_open()**.

SYNOPSIS

```
#include <semaphore.h>
sem_t *sem_open( const char *name,int oflag );
sem_t *sem_open( const char *name,int oflag,
                 mode_t mode,unsigned int counter );
const char *name -nazwa semafora nowego
(lub istniejącego i nowootwieranego)
int oflag -maska bitowa:
O_CREAT-utwórz gdy nie istnieje,
O_EXCL -na wyłączność (jeżeli istnieje O_CREAT|O_EXCL wywoła błąd)
(! stałe O_CREAT oraz O_EXCL zawarte w fcntl.h !)
mode_t mode -maska uprawnień jak w przypadku open() i creat()
unsigned int counter -wartość inicjująca semafor (! unsigned !)
```

RETURN

```
sem_t -identyfikator otwartego semafora
```

ERRORS

```
SEM_FAILED -jeżeli błąd (w bieżącej implementacji jest to 0)
```

W momencie kiedy nie będzie on w danym procesie już do niczego potrzebny należy dokonać zamknięcia **sem_close()**. Jeżeli w danej rodzinie procesów semafor nazwany nie będzie już używany, jego ostatecznego usunięcia dokonuje **sem_unlink()**.

SYNOPSIS

```
#include <semaphore.h>
int sem_close( sem_t *id );
int sem_unlink( const char *name );
sem_t *id -identyfikator semafora (do zamknięcia)
const char *name -nazwa semafora do usunięcia
```

RETURN

0 *-jeżeli operacja zakończona powodzeniem*

ERRORS

-1 *-jeżeli błąd*

O ile wartość licznika semafora inicjujemy w chwili tworzenia, to pobrać możemy przez

SYNOPSIS

```
#include <semaphore.h>
int sem_getvalue( sem_t *id,int *counter );
int *counter -zwracana wartość licznika semafora
(! zauważmy, że inicjująca w sem_open() jest unsigned int a tutaj int !)
```

RETURN

0 *-jeżeli operacja zakończona powodzeniem*

ERRORS

-1 *-jeżeli błąd*

MECHANIZMY SYNCHRONIZACJI IPC

Utwórzmy w takim razie semafor (ogólny) nazwany ***POSIX***, zainicjujemy i odczytamy stan jego licznika.

```
#include <stdio.h>      ...operacje wejścia/wyjścia
#include <sys/stat.h>       ...maski uprawnień (choć niekonieczne)
#include <fcntl.h>      ...stała OCREAT
#include <semaphore.h>  ...semafory POSIX, w ogólności (konieczne)

int main( int argc,char **argv )
{
   sem_t *id;           ...identyfikator tworzonego semafora
   int counter;         ...licznik semafora (pomocniczo)
   char name[] = "km";  ...nazwa semafora (jeżeli nazwany, jakaś musi
być)
                            niektóre implementacje semaforów POSIX
                            wymagają aby nazwa zaczynała się od '/',
                            czyli będzie
                            char name[] = "km";

   counter = 7;         ...inicjujemy zmienną dla licznika
```

MECHANIZMY SYNCHRONIZACJI IPC

```
    ...spróbujmy w takim razie utworzyć/otworzyć semafor (nazwany)
    if((id=sem_open(name,O_CREAT,S_IRUSR|S_IWUSR,(unsigned)counter))!=SEM_FAILED)
    {
       ...jeżeli się udało, to ciąg dalszy
       printf( "*** otwarto semafor\t%s\n",name );

       if( !sem_getvalue( id,&counter ) ) ...odczyt licznika
       { printf( "    licznik semafora\t%d\n",counter ); }
       else{ perror( "*** sem_getvalue()\t" ); }

       if( !sem_close( id ) ){ printf( "*** zamknięto semafor\t%s\n",name ); }
       else{  perror( "*** sem_close()\t" ); }

       /*...na wstępie pozostawmy te linie w komentarzu
       if( !sem_unlink( name ) ){ printf( "*** usunięto semafor\t%s\n",name ); }
       else{   perror( "*** sem_unlink()\t" ); }
       */
    }
    else{ perror( "....sem_open()...." ); } ...cóż, gdyby coś poszło nie tak

    return 0;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

Załóżmy, że kod źródłowy zapisano pod nazwą **named.c**

W odróżnieniu od semaforów ***SysV***, które stanowią integralną część ***IPC*** i funkcji jądra systemu – semafory ***POSIX*** takimi nie są. W przypadku kiedy chcemy z nich korzystać,

❗ kod musi być konsolidowany ❗

z biblioteką ***real-time library***, czyli `librt.so`

albo (zależnie od dystrybucji)

z biblioteką `libpthread.so`

wszak proces jest także wątkiem.

W takim razie (alternatywnie, zależnie od dystrybucji)

```
$ gcc -Wall named.c -o named -lrt
$ gcc -Wall named.c -o named -lpthread
```

a wykonanie

```
$ ./named
*** otwarto semafor     km
    licznik semafora 7
*** zamknięto semafor   km
```

Gdyby wykonać

```
$ ls -l /dev/shm/
razem 4
-rw------- 1 kmirota users  32 cze  2 10:17 sem.km
drwxr-xr-x 2 root    root  200 cze  2 08:16 sysconfig
```

bowiem semafor nazwany jest montowany w węźle **/dev/shm**.

Wywołanie funkcji **sem_unlink()** - tradycyjnie, jak w przypadku wszystkich ***unlink()** - dokonuje usunięcia dowiązania

```
sem.km
```

w **/dev/shm/**.

W takim razie gdyby z wyjściowego kodu źródłowego usunąć komentarz, czyli wykonać ostatnią instrukcję **if(){}else{}**

```
if( !sem_unlink( name ) ){ printf( "*** usunięto semafor\t%s\n",name ); }
else{   perror( "*** sem_unlink()\t" ); }
```

to, po wykonaniu

```
$ ./named
*** otwarto semafor     km
    licznik semafora    7
*** zamknięto semafor   km
*** usunięto semafor    km
```

mamy wynik

```
$ ls - /dev/shm/
razem 0
drwxr-xr-x 2 root root 200 cze  2 08:16 sysconfig
```

a więc usunięto **sem.km**.

MECHANIZMY SYNCHRONIZACJI IPC

Semafory nienazwane nie stanowią dowiązań do systemu plików, a więc użycie nie musi poprzedzać użycie funkcji ***plikowych*** typu ***open()**, **close()** czy – w końcu - ***unlink()**.

Użycie wymaga natomiast wywołań innych funkcji alokujących obszar pamięci dla niego **sem_init()** i zwalniającej **sem_destroy()**.

SYNOPSIS

```
#include <semaphore.h>
int sem_init( sem_t *id,int pshared,unsigned int counter );
int sem_destroy( sem_t *id );
sem_t *id -identyfikator semafora
int pshared -współdzielenie semafora między procesami
0 -dostępny dla wątków procesu, w przeciwnym razie,
także dla innych procesów
unsigned int counter -wartość inicjująca licznik semafora
```

RETURN

```
0 -jeżeli operacja zakończona powodzeniem
```

ERRORS

```
-1 -jeżeli błąd
```

MECHANIZMY SYNCHRONIZACJI IPC

Gdyby skorygować kod z przykładu wcześniejszego, dla wariantu semafora nienazwanego.

```
#include <stdio.h>
#include <semaphore.h>
int main( int argc,char **argv )
{
   sem_t id;
   int counter;

   counter = 7;
   if( !sem_init( &id,0,counter ) )
   {
      printf( "*** inicjacja semafor\n" );
      if( !sem_getvalue( &id,&counter ) )
      { printf( "    licznik semafora\t%d\n",counter ); }
      else{ perror( "*** sem_getvalue()\t" ); }
      if( !sem_destroy( &id ) ){ printf( "*** usunięcie semafor\n" ); }
      else{  perror( "*** sem_destroy()\t" ); }
   }
   else{ perror( "....sem_init()...." ); }

   return 0;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

Jeżeli kod źródłowy zapiszemy pod nazwą **unnamed.c**, to jego kompilacja

```
$ gcc -Wall unnamed.c -o unnamed  -lrt
```

oczywiście musimy pamiętać o konsolidacji z właściwą biblioteką, a wykonanie

```
$ ./unnamed
*** inicjacja semafor
licznik semafora    7
*** usunięcie semafor
```

Akcje **P()** i **V()**, czyli **wait()** i **signal()** realizowane są wywołaniami

SYNOPSIS

```
#include <semaphore.h>
int sem_wait( sem_t* id );
int sem_trywait( sem_t* id );
int sem_post( sem_t* id );
```

RETURN

0 *-jeżeli operacja zakończona powodzeniem*

ERRORS

-1 *-jeżeli błąd*

Pierwsza z funkcji – **sem_wait()** - powoduje dekrementację licznika semafora, jeżeli jego wartość jest większa od zera to proces będzie kontynuował, w przeciwnym razie zostanie zablokowany (**sem_trywait()**, generuje zamiast tego błąd). Druga zaś powoduje inkrementację, jeżeli wartość licznika stanie się większa od zera, to proces zablokowany na nim będzie obudzony (o ile taki jest).

MECHANIZMY SYNCHRONIZACJI IPC

Właściwie użycie semaforów ***POSIX*** w relacji do ***SysV*** nie różni się istotnie (jeżeli pamiętać będziemy o różnicach podanych na wstępie). Poniżej prosty przykład.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <semaphore.h>

int main( int argc,char** argv )
{
   sem_t *id;
   pid_t child;
   int status;

   id = sem_open( "szlaban",O_CREAT,S_IRUSR|S_IWUSR,0 );
   switch( (int)(child=fork()) )
   {
      case -1: perror("...fork()..."); exit( 1 ); break;
```

```
    case 0: ...kod dla potomka
    printf("*** [%u] potomek czeka na semafor (%p)\n",(unsigned)getpid(),id);
    sem_wait( id );
    printf( "*** [%u] potomek zakończył\n",(unsigned)getpid() );
    exit( 0 );
    default:  ...kod dla procesu nadrzędnego
    printf( "*** [%u] ustawia semafor (%p)\n",(unsigned)getpid(),id );
    sem_post( id );
    if( !wait( &status ) ){ perror( "... wait()..." ); exit( 2 ); }
    else
    {
       printf( "*** [%u] wygląda, że to wszystko\n",(unsigned)getpid() );
       sem_close( id ); sem_unlink( "szlaban" );
    }
    }

   return 0;
}
```

Wykonanie

```
$ ./forky
*** [9691] potomek czeka semafor (0x2b1c790bc000)
*** [9690] ustawia semafor (0x2b1c790bc000)
*** [9691] potomek zakończył
*** [9690] wygląda, że to wszystko
```

MECHANIZMY SYNCHRONIZACJI IPC

O ile semafory kojarzone są raczej z procesami, to nic nie stoi – formalnie – na przeszkodzie aby użyć je wobec wątków. W kolejnym przykładzie użyjemy semafora nienazwanego (***POSIX***) celem zabezpieczenie wyłączności wykonania pewnego bloku kodu funkcji wątku. Załóżmy funkcję wątku postaci

```
void* thread( void *ptr )
{
    int i = *((int *) ptr); ...z argumentu pobierzemy numer kolejny wątku
    printf( "wątek %d: start\n",i );

    sem_wait( &id );     ...wątek wykonuje P(), wyłączność
    printf( "wątek %d: krytyczna, start\n",i );
    printf( "wątek %d: n = %d\n",i,n );
    printf( "wątek %d: n++\n",i );    n++;
    printf( "wątek %d: n = %d\n",i,n );
    printf( "wątek %d: krytyczna, stop\n",i );
    sem_post( &id );     ...wątek wykonuje V(), zwolnienie

    printf( "wątek %d: stop\n",i );
    pthread_exit( 0 );
}
```

Działanie wątku polega na pobraniu wartości pewnej zmiennej globalnej **n** jej inkrementacji.

MECHANIZMY SYNCHRONIZACJI IPC

Kod programu przedstawia się następująco.

```
#include <stdio.h>          ...poniekąd "zestaw standardowy" plików włączanych
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#define SHARE  1            ...współdzielenie semafora między procesami (true)
sem_t id;                   ...identyfikator używanego semafora
int n;                      ...zmienna globalna na której działać będą wątki
int main( int argc,char** argv )
{
   int i[]={1,2};               ...tablica numerów wątków
   pthread_t first,second;
   sem_init( &id,!SHARE,1 );  ...tworzymy semafor (POSIX, nienazwany)
   ...wątki startują
   pthread_create( &first,  NULL,thread,(void *)(i+0) );
   pthread_create( &second, NULL,thread,(void *)(i+1) );
   /* tutaj pracują wątki */
   pthread_join( first,NULL ); pthread_join(second,NULL );
   ...wątki zakończyły działanie
   sem_destroy( &id );         ...semafor usunięty
   return 0;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

Oczywiście trzeba pamiętać o konsolidacji z właściwymi bibliotekami

```
$  gcc -Wall threads.c -o threads -lpthread -lrt
```

lub tylko

```
$  gcc -Wall threads.c -o threads -lpthread
```

a wykonanie

```
$ ./threads
wątek 1: start ----------------------
wątek 1: krytyczna, start
wątek 1: n = 0
wątek 1: n++
wątek 1: n = 1
wątek 1: krytyczna, stop
wątek 1: stop -----------------------
wątek 2: start ----------------------
wątek 2: krytyczna, start
wątek 2: n = 1
wątek 2: n++
wątek 2: n = 2
wątek 2: krytyczna, stop
wątek 2: stop -----------------------
```