MECHANIZMY SYNCHRONIZACJI IPC

Kolejnym elementem funkcjonalnym ***InterProcess Communication (IPC) UNIX System V*** a także ***POSIX***[*] są semafory. Stanowią one realizację najwcześniejszych pomysłów i sposobów rozwiązania problemów współużytkowania zasobów i koordynacji, który zaproponował ***Edsger Wybe Dijkstra***.

Podobnie jak w przypadku wszystkich obiektów IPC, z poziomu interface systemowego użytkownika można się do nich odwołać za pomocą komendy **ipcs**.

```
$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xcbc384f8 163841     kmirota    600        1
```

Każda z nich identyfikowana jest unikalnym kluczem (**semkey**) oraz identyfikatorem (**semid**), każda posiada określonego właściciela (tutaj: **kmirota**) i prawa dostępu (**perms**). Ostatnia kolumna podaje ilość semaforów w tablicy (tutaj: **1**). Usunięcie takiego obiektu z pamięci jądra odbywa się za pomocą **ipcrm [-S semkey | -s semid]**, czyli, tutaj mogłoby to być

```
$ ipcrm -S 0xcbc384f8
```

albo

```
$ ipcrm -s 163841
```

[*] Przedstawione w tu rozwiązania odnoszą się de facto do ***UNIX System V*** nie zaś ***POSIX***. Ten ostatni wprowadził dodatkowo inny sposób zarządzania semaforami.

MECHANIZMY SYNCHRONIZACJI IPC

Jądro systemowe zarządza tablicą semaforów wykorzystując strukturę danych o następującej definicji zawartej w **sys/sem.h** (a de facto włączanej z **bits/sem.h**)

```
struct semid_ds
{
    struct ipc_perm sem_perm;      /* struktura opisująca uprawnia */
    __time_t sem_otime;            /* czas ostatniego semop()      */
    __time_t sem_ctime;            /* czas ostatniego semctl()     */
    unsigned long int sem_nsems;   /* ilość semaforów w tablicy    */
    /* inne, odpowiednio do potrzeb implementacji                  */
};
```

Generalnie obligatoryjnymi są 4 wymienione pola struktury **semid_ds**, a standard ***UNIX System V*** i ***POSIX***, dopuszcza rozszerzania tej definicji. Uprawnienia przechowywane w – podobnie jak i dla pozostałych obiektów ***IPC*** – w strukturze **ipc_perm** (**bits/ipc.h**).

```
struct ipc_perm
{
    __key_t __key;   __uid_t uid;
    __gid_t gid;     __uid_t cuid;
    __gid_t cgid;    unsigned short int mode;
    unsigned short int __seq;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

Pierwszą czynnością będzie oczywiście uzyskanie dostępu lub utworzenie (o ile nie istnienie) do określonego zbioru (tablicy) semaforów a ściślej do struktury typu **semid_ds**.

SYNOPSIS

```
#include <sys/sem.h>
int semget( key_t key,int nsems,int semflg );
key_t key klucz identyfikujący zbiór semaforów,
jeżeli użyjemy IPC_PRIVATE to będzie prywatna o kluczu automatycznym
int nsems ilość max semaforów w danej tablicy
int semflg maska bitowa sposobu tworzenia tablicy semaforów
```

RETURN

```
indeks semid do struktury semid_ds
zarządzającej danym zbiorem semaforów
```

ERROR

```
-1
```

Maska tworzenie zbioru semaforów posiada zwykle postać sumy bitowej

```
semflag = IPC_CREAT | IPC_EXCL | mask
```

IPC_CREAT żądanie utworzenie lub uzyskanie dostępu

IPC_EXCL żądanie wyłącznego zbioru (samodzielnie nie ma sensu), ale jeżeli istnienie to zostanie wygenerowany błąd

mask maska uprawnień, podobnie jak dla ogółu obiektów *IPC* (**9** młodszych bitów)

MECHANIZMY SYNCHRONIZACJI IPC

Zgodnie z ***POSIX.1-2001*** wartości dla wszystkich nowotworzonych semaforów są nieokreślone, aczkolwiek w przypadku wielu implementacji systemowych – mimo wszystko wprowadza się inicjowanie (i tak przykładowo w ***LINUX*** są inicjowane zerami `0`). Do dobrej praktyki należy zakładanie iż wartość jest ta jest nieokreślona.

Warto jeszcze zauważyć, że w odniesieniu do :

- maksymalna ilość semaforów związana z danym identyfikatorem jest ograniczona predefiniowanym w `linux/sem.h` parametrem `SEMMSL`;
- wartość ta określana jest w momencie tworzenia tablicy i nie może być później zmieniana
- jeżeli odwołujemy się do istniejącej już tablicy, to wartość ta nie ma znaczenia może być zero (`0`).

Podając wartość klucza można skorzystać z funkcji

```
key_t ftok( const char *pathname,int id );
```

podając pewne arbitralnie obrane: `pathname` ścieżką (ale istniejącą) oraz `id` będący liczbą całkowitą niezerową.

MECHANIZMY SYNCHRONIZACJI IPC

Przygotujemy w takim razie program, który utworzy zbiór (tablicę jądra) semaforów o zadany (w linii komendy) rozmiarze. Z założenia tablica ta będzie dostępna dla procesów użytkownika.

```
#include <stdio.h>      //...standardowe wejście/wyjście
#include <stdlib.h>     //...komunikat błędu perror()
#include <unistd.h>     //...identyfikacja procesu geppid()
#include <sys/sem.h>//...stąd deklaracje dot. semaforów
#include <sys/stat.h>   //...maski uprawnień, choć można inaczej

//skoro czytamy z linii komend, to nagłówek main() ...
int main( int argc,char** argv )
{

   int nsems;    //...zmienna dla ilość semaforów w tworzonym zbiorze
   int semflag;  //...zmienna dla maski tworzenia semaforów
   key_t key;    //...zmienna dla klucza identyfikującego zbiór
```

MECHANIZMY SYNCHRONIZACJI IPC

```
    if( argc>1 ) //...sprawdźmy na początek, czy wywołanie było poprawne
    {
        sscanf( argv[1],"%d",&nsems ); //...odczytujemy ilość semaforów
        if( nsems>0 ) //...sprawdźmy, tak na wszelki wypadek
        {
            key = ftok( "/tmp",'a'+'t'+'h'+'r'+'i'+'r' );
            //...taki sobie komunikat diagnostyczny
            printf(   "[pid=%u] tworzy zbiór %d semaforów
                      [key=%x]\n",(unsigned)getpid(),nsems,(unsigned)key );
            //...przygotujemy maskę tworzenia semaforów
            semflag = IPC_CREAT | S_IRUSR | S_IWUSR;
            //...i to już wszystkie czynności przygotowawcze do semget()
            if( semget( key,nsems,semflag )==-1 ) //...jeżeli,to kłopot
            { perror( "\tsemget()..." ); exit( 3 ); }
        /*- a dalej to już tylko diagnostyka ewentualnych błędów -*/
        }
        else{printf("\tbłędna ilość semaforów (n=%d)\n",nsems );exit(2);}
    }
    else
    {   printf( "\t%s [%s]\n%s",argv[0],"n",
        "\tn -rozmiar tworzonej tablicy semaforów\n" ); exit( 1 ); }
    return 0;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

Kompilacja i konsolidacja, jeżeli przyjąć że program zapisano pod nazwą **setsem.c**

```
$ gcc -Wall setsem.c -o setsem
```

Na początek sprawdźmy jakie tablice semaforów mamy zdefiniowane, czyli

```
$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xcbc384f8 294912     kmirota    600        1
0x0056a4d5 327681     kmirota    660        1
```

akurat mamy dwie, w bieżącej chwili - powiedzmy, że teraz dodamy tablicę 7 semaforów

```
$ ./setsem 7
[pid=7313] tworzy zbiór 7 semaforów [key=8a060481]
```

gdyby ponownie sprawdzić zdefiniowane tablice semaforów

```
$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xcbc384f8 294912     kmirota    600        1
0x0056a4d5 327681     kmirota    660        1
0x8a060481 458754     kmirota    600        7
```

czyli pojawiła się i nasza (**0x8a060481**), która może zawierać do 7 semaforów (max).
Gdyby chcieć ją usunąć, to

```
$ ipcrm -S 0x8a060481
```

lub też (co przyniesie identyczny skutek)

```
$ ipcrm -s 458754
```

MECHANIZMY SYNCHRONIZACJI IPC

Sterowanie tablicą semaforów jak i jej zawartością odbywa się po przez **semctl()**.

SYNOPSIS

```
#include <sys/sem.h>
int semctl( int semid,int semnum,int cmd, ... );
int semid identyfikator zbioru (tablicy) semaforów,
          której odwołanie dotyczy
int semnum numer semafora w danym zbiorze (tablicy),
          którego odwołanie dotyczy
int cmd   rodzaj działania, komenda
...       czwarty argument jest opcjonalny i zależny od cmd
```

RETURN

```
GETNCNT      wartość semncnt
GETPID    wartość sempid.
GETVAL    wartość semval.
GETZCNT      wartość semzcnt.
IPC_INFO  indeks do max wartości w tablicy jądra semaforów
          (tylko LINUX)
SEM_INFO  identycznie jak IPC_INFO.
SEM_STAT  identyfikator semafora o danym indeksie semid.
0   w pozostałych przypadkach, jeżeli sukces
```

ERROR

```
-1
```

MECHANIZMY SYNCHRONIZACJI IPC

Zdefiniowano następujące działania **cmd** (trzeci parametr formalny **semctl()**):

IPC_STAT odczyt/zapis zawartości struktury **semid_ds** dla podanego **semid**

IPC_SET (**semnum** jest tutaj ignorowane)

IPC_RMID natychmiastowe usunięcie zbioru semaforów skojarzonych z **semid**, (procesy zablokowane są wznowione, **semnum** jest ignorowane)

IPC_INFO odczyt ogólnych informacji systemowych, zwracane w strukturze

```
struct  seminfo
{
    int semmap;  /* aktualnie nie używane */
    int semmni;  /* max ilość zbiorów semaforów */
    int semmns;  /* max ilość semaforów we wszystkich zbiorach */
    int semmnu;  /* aktualnie nie używane */
    int semmsl;  /* max ilość semaforów w zbiorze */
    int semopm;  /* max ilość operacji dla semop() */
    int semume;  /* aktualnie nie używane */
    int semusz;  /* wykorzystywana przy odtwarzaniu (undo) */
    int semvmx;  /* max wartość semafora */
    int semaem;   /* wykorzystywana przy odtwarzaniu (undo) */
};
```

MECHANIZMY SYNCHRONIZACJI IPC

GETALL	wartości (stany) dla wszystkich semaforów w danej tablicy podanej za pomocą wskazania, jako 4 parametr **semctl()** (**semnum** jest ignorowane)
GETNCNT	**semncnt**, czyli ilość procesów zablokowanych danym semaforem, i czekających na zwiększenie jego wartości
GETPID	*PID* procesu, który zmodyfikował semafor **semnum** za pomocą **semop()**
GETVAL	odczyt wartości **semval** dla danego **semnum** (**unsigned short semval**)
GETZCNT	ilość procesów **semzcnt**, które na danym semaforze czekają aż osiągnie **0**
SETALL	ustawienia wartości **semval** dla wszystkich semaforów tablicy
SETVAL	ustawienia wartości **semval** dla danego semafora **semnum**

Czwarty, opcjonalny, parametr **semctl()** ma postać unii pól

```
union semun
{
    int value;                  //...wartość dla SETVAL
    struct semid_ds *stat;      //...bufor dla IPC_STAT, IPC_SET
    unsigned short int *array; //...tablica dla GETALL, SETALL
    struct seminfo *info;       //...bufor dla IPC_INFO
};
```

której właściwe pole wybierane jest odpowiednio do rodzaju działań.

Jej definicja musi być wprowadzona we własnym zakresie.

MECHANIZMY SYNCHRONIZACJI IPC

Gdyby w takim razie – nawiązując do wcześniejszego przykładu – chcieć usunąć zdefiniowany uprzednio zbiór semaforów, należałoby zmodyfikować jak niżej.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/sem.h>
#include <sys/stat.h>

int main( int argc,char** argv )
{

   int nsems,semflag,semid;
   key_t key;
    if( argc>1 )
    {
       sscanf( argv[1],"%d",&nsems );
```

MECHANIZMY SYNCHRONIZACJI IPC

```
if( nsems>0 )
{
    key = ftok( "/tmp",'a'+'t'+'h'+'r'+'i'+'r' );
    printf(   "[pid=%u] tworzy zbiór %d semaforów  [key=%x]\n",
              (unsigned)getpid(),nsems,(unsigned)key );
    semflag = IPC_CREAT | S_IRUSR | S_IWUSR;
    semid = semget( key,nsems,semflag );
    if( semid==-1 ){ perror( "\tsemget()..." ); exit( 3 ); }
    else
    {
        if( semctl( semid,0x0,IPC_RMID) == -1)
        { perror("\tsemctl()"); exit(1); }
    }
}
else
{
    ... itd., jak wcześniej
```

MECHANIZMY SYNCHRONIZACJI IPC

W efekcie, na wstępie mamy

```
$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

i wywołanie programu

```
$ ./semrm 7
[pid=15192] tworzy zbiór 7 semaforów [key=8a060481]
```

a w efekcie

```
$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

MECHANIZMY SYNCHRONIZACJI IPC

Operacje na semaforach wykonuje się za pomocą funkcji **semop()**.

SYNOPSIS

```
#include <sys/sem.h>
int semop( int semid,struct sembuf *sops,unsigned nsops );
int semid identyfikator tablicy semaforów
struct sembuf *sops wskazanie do tablicy struktur wartości semafora
    struct sembuf
    {
        ushort_t  sem_num;  /* numer semafora */
        short     sem_op;   /* operacja na semaforze, kod */
        short     sem_flg;  /* znaczniki dla operacji */
    };
unsigned nsops ilość elementów w tablicy struktur sops w tablicy
```

RETURN

0 *gdy operacja zakończona sukcesem*

ERROR

-1

MECHANIZMY SYNCHRONIZACJI IPC

Zasadniczo rodzaj akcji jaka podjęta będzie wobec konkretnego semafora tablicy, determinowany jest de facto sposobem zainicjowania pól struktury

```
struct sembuf { ushort_t  sem_num; short sem_op; short sem_flg; };
```

Znaczenie pierwszego z nich – **sem_num** - jest oczywiste, określa indeks w tablicy semaforów. Dla trzeciego pole – **sem_flag** – rozpoznawalne są dwie maski bitowa dane stałymi predefiniowanymi

IPC_NOWAIT	*wykonaj jako operację nieblokującą*
SEM_UNDO	*efekt działań na semaforze będzie anulowany w momencie kiedy proces zakończy działanie*

trzecia z możliwości, to oczywiście

0x0	*czyli działanie domyśle, odwołanie blokujące*

Interpretacja ostatniego parametru uzależnione jest od jego znaku, wykonane będzie:

V() *czyli* signal(), *jeżeli* sem_op > 0

P() *czyli* wait(), *jeżeli* sem_op < 0

natomiast, kiedy parametr będzie miał wartość zerową, to sprawdzone będzie

czy semval == 0 ? (wait-for-zero)

jeżeli

true *proces kontynuuje*

true *proces zawiesza swoje działanie i czeka aż semafor osiągnie* 0

false i IPC_NOWAIT (*czyli bez blokowania*) *wygenerowany błąd* semop()

MECHANIZMY SYNCHRONIZACJI IPC

Utwórzmy teraz krótki program w którym zdefiniujemy pojedynczy semafor, zainicjujemy jego wartość a następnie odwołamy się do niego z procesu potomnego wygenerowanego za pomocą `fork()`.

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

int main( void )
{
    int status;
    key_t semkey;                //...  zmienne opisujące
    int nsems,semflag,semid;     //     tworzoną tablicę semaforów
    struct sembuf sems;          //...  struktura dla semop()

    nsems   = 1; //... tworzymy tablicę semaforów, z jednym semaforem
    semkey  = ftok( "/tmp",'k'+'m' );
    semflag = IPC_CREAT | S_IRUSR | S_IWUSR;
    semid   = semget( semkey,nsems,semflag );
```

```
    sems.sem_num = 0;               //... indeks do semafora, pierwszy=0 !
    sems.sem_op = 7;                //... może współdzielić 7 procesów
    sems.sem_flg = 0x0;             //... czyli z blokadą
    semop( semid,&sems,nsems );     //... inicjujemy semafor
    //... teraz pozostaje już tylko utworzyć proces potomny
    switch( fork() )
    {
       case -1: //... obsługa błędu fork()
          printf( "!.!.!... błąd fork()...!.!.!\n" ); exit( 1 ); break;
       case  0: //... kod dla procesu potomnego
          printf( "[%u] semval=%d\n",(unsigned)getpid(),
             semctl( semid,0,GETVAL ) ); exit( 0 ); break;
       default: //...kod dla procesu nadrzędnego
          wait( &status );
          printf(   "[%u] semval=%d\n",(unsigned)getpid(),
                    semctl( semid,0,GETVAL ) );
          semctl( semid,0x0,IPC_RMID); //... usuwamy semafory
    }

    return 0;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

Wykonanie programu powinno przynieść rezultat

```
$./semval
[17887] semval=7
[17886] semval=7
```

Wniosek stąd, że prawidłowo zainicjowaliśmy semafor i zarówno proces macierzysty jak i potomek prawidłowo odwołują się do niego.

W tym momencie warto zadać sobie pytanie jak to jest możliwe, że

potomek ma pełną informację o semaforze utworzonym przed wywołaniem `fork()` ?

w końcu stanowi on niezależną instancję programu.

Odpowiedź jest bardzo prosta a poniekąd oczywista i wynika z własności funkcji **`fork()`**:

obiekty IPC, utworzone przed wywołaniem `fork()`***, są dziedziczone***

Zauważmy, że w przeciwnym przypadku ***IPC*** jako narzędzie komunikacji międzyprocesowej miałoby bardzo ograniczoną funkcjonalność (a i sens).

MECHANIZMY SYNCHRONIZACJI IPC

Prześledźmy teraz poniższy kod w którym semafor będzie użyty celem ochrony sekcji krytycznej kolejnych procesów.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/sem.h>
union semun //...struktura potrzebna dla semctl()
{
   int value; struct semid_ds *stat;
   unsigned short int *array; struct seminfo *info;
};

int main( int argc,char** argv )
{
    key_t key;
    int semid,flag;
    struct sembuf P={ 0,-1,0}; //...żądanie dostępu   {numer,operacja,flaga}
    struct sembuf V={ 0,+1,0}; //...zwolnienie zasobu {numer,operacja,flaga}
    union semun control;
    int i,p,k,n;
    double x;
```

MECHANIZMY SYNCHRONIZACJI IPC

```
if( argc<2 ) //...sprawdźmy, czy aby wywołanie jest poprawne
{
    printf( "%s %s\n  %s\n",argv[0],"[n]","n-krotność wykonania procesu" );
    exit( 1 );
}
sscanf( argv[1],"%d",&p ); //...jeżeli tak, to czytamy ilość powtórzeń

key = ftok( "/tmp", 'k'+'m' ); flag = IPC_CREAT | S_IRUSR | S_IWUSR;
semid = semget( key,1,flag); //... tworzymy tablicę semaforów (pojedynczy)
control.value = 1; //...inicjowanie semafora (binarnego) wartością 1
semctl( semid,0x0,SETVAL,control );

switch( fork() ) //...utworzenie procesu potomnego
{
    case -1: //...obsługa błędu fork
        printf( "!.!.!...fork()...!.!.!\n" ); exit( 1 ); break;
    case  0: //...kod dla procesu potomnego
        printf( "...[%u]...proces potomny.......start\n",(unsigned)getpid());
        fflush( stdout ); break;
    default://...kod dla procesu macierzystego
        printf( "...[%u]...proces macierzysty...start\n",(unsigned)getpid());
        fflush( stdout ); break;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

A teraz **p**-krotne wykonania sekcji krytycznych procesów.

```
    for( i=0;i<p;i++ )
    {
        semop( semid,&P,1 );
        //...początek sekcji krytycznej
        printf("   [%u]...rozpoczyna wykonywanie sekcji krytycznej\n",
               (unsigned)getpid());
        n = rand();
        for( k=0,x=0;k<n;k++ ){ x += (double)rand()/RAND_MAX; }
        printf( "   n=%d x=%f\n",n,x/(double)n ); fflush( stdout );
        printf("   [%u]...kończy wykonywanie sekcji krytycznej\n",
               (unsigned)getpid() );
        //...koniec sekcji krytycznej
        semop( semid,&V,1 );
    }

    semctl( semid,0x0,IPC_RMID,control ); //...usuwamy tablicę semaforów

    return 0;
}
```

Wykonanie przedstawia się następująco

```
$ ./forks-1 2
...[9939]...proces potomny.......start
   [9939]...rozpoczyna wykonywanie sekcji krytycznej
...[9938]...proces macierzysty...start
   n=1804289383 x=0.499997
   [9939]...kończy wykonywanie sekcji krytycznej
   [9938]...rozpoczyna wykonywanie sekcji krytycznej
   n=1804289383 x=0.499997
   [9938]...kończy wykonywanie sekcji krytycznej
   [9939]...rozpoczyna wykonywanie sekcji krytycznej
   n=298072528 x=0.500004
   [9939]...kończy wykonywanie sekcji krytycznej
   [9938]...rozpoczyna wykonywanie sekcji krytycznej
   n=298072528 x=0.500004
   [9938]...kończy wykonywanie sekcji krytycznej
```

Niezależnie od czasu trwania wykonania sekcji krytycznej, jest ona w każdym przypadku skutecznie chroniona, choć znajduje się w obrębie części wspólnej kodu procesów. Zwróćmy także uwagę na kolejność wykonywania się procesów.

MECHANIZMY SYNCHRONIZACJI IPC

Często zależeć nam będzie na uzyskaniu pożądanej kolejności wykonania procesów. Można to łatwo osiągnąć za pomocą semaforów.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/sem.h>

int main( void )
{
    union semun
    {
        int value; struct semid_ds *stat;
        unsigned short int *array; struct seminfo *info;
   } control;
   key_t key;
   int flag,semid;
   struct sembuf sems;

   key = ftok( "/tmp",'k'+'m' );  flag = IPC_CREAT | S_IRUSR | S_IWUSR;
   semid = semget( key,1,flag);
   control.value =+1;  semctl( semid,0x0,SETVAL,control );
```

```
    sems.sem_num = 0;  sems.sem_flg = 0x0;
    switch( (int)fork() )
    {
        case -1: perror( "...fork()...\t" ); exit( 1 ); break;
        case  0:
            sems.sem_op =-1; semop( semid,&sems,1 );
            printf( "...child...\t:%u\n",(unsigned)getpid() );
            sems.sem_op =+2; semop( semid,&sems,1 );
            break;
        default:
            sems.sem_op =-2; semop( semid,&sems,1 );
            printf( "...master...\t:%u\n",(unsigned)getpid() );
            sems.sem_op =+1; semop( semid,&sems,1 );
            break;
    }
    semctl( semid,0x0,IPC_RMID );

    return 0;
}
```

Z racji sposobu zainicjowania i korzystania z semafora, proces potomny zawsze będzie wykonany jako pierwszy, a dopiero kiedy on zakończy działanie, proces nadrzędny odzyska sterowanie.

MECHANIZMY SYNCHRONIZACJI IPC

Posługując się semaforami można również wymusić pewną krotność wykonania określonego procesu podczas gdy wykonanie innego czasowo zawieszamy.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/sem.h>

int main( void )
{
    union semun   //...tę strukturę musimy zdefiniować we własnym zakresie
    {
        int count; struct semid_ds *stats;
        unsigned short int *array;struct seminfo *infos;
    } control;

    //...struktury wykorzystywane przez semop()
    struct sembuf P0={ 0,-1,0 },V0={ 0,1,0 },Z0={ 0,0,0 };
    struct sembuf P1={ 1,-1,0 },V1={ 1,1,0 },Z1={ 1,0,0 };
    //...i cała reszta
    key_t key; pid_t pid;
    int semid,flag,nsems,step;
```

MECHANIZMY SYNCHRONIZACJI IPC

```
key=ftok( "/tmp", 'k'+'m' );flag=IPC_CREAT | S_IRUSR | S_IWUSR; nsems = 2;
semid = semget( key,nsems,flag );
control.count = 1; semctl(semid,0,SETVAL,control);
control.count = 5; semctl(semid,1,SETVAL,control); //czyli będzie 5 cykli

pid = fork(); //...uaktywniamy proces potomny
for( step=0;step<control.count;step++ )
{
    if( !pid ) //...to wyłacznie dla potomka
    {
        printf( "[%u]...procesu potomny...start\n",(unsigned)getpid() );
        semop( semid,&P0,1 ); //...ustawiamy semafor '0'
        printf( "[%u]...krytyczna...start\n",(unsigned)getpid() );
        sleep( 1 );
        printf( "[%u]...krytyczna...stop\n",(unsigned)getpid() );
        semop( semid,&V0,1 ); //...i zwalniamy semafor '0'
        //...jeszcze zwiększymy o 1 wartość semafora 1,
        //   blokującego proces nadrzędny
        semop( semid,&P1,1 ); //...zwiększamy wartość o '1' dla
        printf( "[%u]...procesu potomny...stop\n\n",(unsigned)getpid() );
    }
} //...no i załóżmy, że to wszystko, co miał zrobić potomek/potomkowie
```

```
    /* Jeżeli pozostawimy ten fragment w komentarzu, to...
    if( !pid )
    {
        printf( "[%u]...proces potomny zwalnia pamięć\n",(unsigned)getpid());
        exit( 0 );
    }
    ...potomek wykona i całą resztę kodu (kiedy zakończy pętlę)*/
    semop( semid,&Z1,1); //...na tym semaforze zatrzymał się parent

    //...przechodzi go w momencie kiedy, właściwą wartość ustawi potomek
    printf( "[%u]...proces nadrzędny odzyskał sterowanie\n", (unsigned) getpid());
    //...na koniec usuwany semafor z pomięci
    semctl( semid,0x0,IPC_RMID );

     return 0;
}
```

MECHANIZMY SYNCHRONIZACJI IPC

Wykonanie przedstawia się następująco:

```
[20844]...procesu potomny...start
[20844]...krytyczna...start
[20844]...krytyczna...stop
[20844]...procesu potomny...stop

[20844]...procesu potomny...start
[20844]...krytyczna...start
[20844]...krytyczna...stop
[20844]...procesu potomny...stop

[20844]...procesu potomny...start
[20844]...krytyczna...start
[20844]...krytyczna...stop
[20844]...procesu potomny...stop

[20844]...procesu potomny...start
[20844]...krytyczna...start
[20844]...krytyczna...stop
[20844]...procesu potomny...stop

[20844]...proces nadrzędny odzyskał sterowanie //...efekt komentarza !
[20843]...proces nadrzędny odzyskał sterowanie
```