Uniwersytet Bielsko-Bialski

LABORATORIUM

Obliczeń Równoległych i Systemów Rozproszonych

Sprawozdanie nr 5

Pamięć Współdzielona IPC

GRUPA: 2B / SEMESTR: 5 / ROK: 3

Igor Gawłowicz / 59096

Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z mechanizmami IPC (InterProcess Communication), szczególnie z pamięcią współdzieloną (shared memory) w systemie UNIX. W trakcie ćwiczenia skupiono się na:

- Użytkowanie poleceń interfejsu użytkownika do wyświetlania informacji o zdefiniowanych obiektach IPC (ipcs -m), oraz usuwania tych obiektów za pomocą ipcrm.
- Zapoznanie z operacjami i strukturami związanymi z zarządzaniem segmentami pamięci współdzielonej: struct shmid_ds, struct ipc_perm.
- Zastosowanie funkcji takich jak shmget(), shmctl(), shmat(), shmdt() do tworzenia, zarządzania oraz usuwania segmentami pamięci współdzielonej.
- Wykorzystanie pamięci współdzielonej do przechowywania i dostępu do danych między procesami.

Przebieg ćwiczenia

Następnym elementem wchodzącym w skład **IPC (InterProcess Communication)**, będzie współbieżność procesów/wątków, często jest powiązana z okresową/asynchroniczną wymianą komunikatów. Po raz pierwszy został wprowadzony w **UNIX System V** w postaci **pamięci (współ-)dzielonej** z angielskiego **shared memory**.

Podobnie jak i w przypadku pozostałych obiektów IPC, z poziomu interface'u użytkownika dostępne są polecenia:

wyświetlające informacje aktualnie zdefiniowanych obiektach IPC

```
$ ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
```

ponieważ obiekty IPC nie są automatycznie usuwane z pamięci operacyjnej, nawet w sytuacji kiedy procesy je tworzące czy wykorzystujące przestały już istnieć – stąd i polecenie

```
$ ipcrm [ -M key | -m id ]
```

umożliwiające usunięcie obszaru pamięci współdzielonej określonego kluczem key albo identyfikatorem id, w pierwszym lub drugim wariancie składni polecenia.

W momencie tworzenia segmentu dzielonego jądro systemowe przydziela dla jego obsługi, dedykowany obiekt danych

```
struct shmid_ds
{
struct ipc_perm shm_perm; /* Uprawnienia do segmentu */
size_t shm_segsz; /* Rozmiar segmentu w bajtach */
time_t shm_atime; /* Ostatnie przyłączenie segmentu */
```

```
time_t shm_dtime; /* Ostatnie odłączenie segmentu */
time_t shm_ctime; /* Ostatnia zmiana */
pid_t shm_cpid; /* PID twórcy */
pid_t shm_lpid; /* PID ostatniego shmat()/shmdt() */
shmatt_t shm_nattch; /* Ilość przyłączonych */
 ... /* ... i ewentualnie jeszcze inne */
};
```

```
struct ipc_perm
{
key_t key; /* key podany w shmget() */
uid_t uid; /* UID właściciela */
gid_t gid; /* GID właściciela */
unsigned short mode;/* uprawnienia + flagi (SHM_DEST,SHM_LOCKED) */
unsigned short seq; /* numer sekwencyjny */
};
```

Proces składa żądanie alokacji lub uzyskania dostępu do istniejącego segmentu pamięci współdzielonej wywołaniem funkcji shmget().

Warto pamiętać, że wywołania funkcji zarządzania procesami skutkują następująco:

- fork() proces potomny dziedziczy przyłączone uprzednio segmenty
- exec() wykonuje przyłączony proces
- exit() wszystkie przyłączone segmenty są zwalniane, choć nie są usuwane

Maska bitowa konstruowana jest operatorem alternatywy bitowej '|', więc bity nie ustawiane inicjujemy zerami. Uprawnienia execute nie mają zastosowania, więc zawsze będzie '0'.

```
user group other BIN OCT DEC HEX
r w x r w x r w x
1 0 ø 0 0 ø 0 0 ø 100000000 0400 256 0x100
0 1 ø 0 0 ø 0 0 ø 010000000 0200 128 0x80
1 1 ø 0 0 ø 0 0 ø 110000000 0600 384 0x180
user group other
r w x r w x r w x
0 0 ø 1 0 ø 0 0 ø 000100000 040 32 0x20
0 0 ø 0 1 ø 0 0 ø 000010000 020 16 0x10
0 0 ø 1 1 ø 0 0 ø 000110000 060 48 0x30
user group other
r w x r w x r w x
0 0 ø 0 0 ø 1 0 ø 000000100 04 4 0x4
0 0 ø 0 0 ø 0 1 ø 000000010 02 2 0x2
0 0 ø 0 0 ø 1 1 ø 000000110 06 6 0x6
user group other
r w x r w x r w x
1 0 ø 1 0 ø 1 0 ø 100100100 0444 292 0x124
```

```
0 1 ø 0 1 ø 0 1 ø 010010010 0222 146 0x92
1 1 ø 1 1 ø 0 1 ø 110110110 0666 438 0x1B6
```

Pobranie informacji o wybranym segmencie pamięci współdzielonej a także niektóre działanie sterujące dostępne są za pośrednictwem funkcji shmctl().

Dopuszczalne są trzy podstawowe operacje cmd za pomocą shmctl()

- IPC_STAT, pobiera informacje o segmencie i kopiuje do struktury struct shmid_ds *buffer
- IPC_SET, kopiuje informacje podane w parametrze struct shmid_ds *buffer do struktury systemowej
- IPC_RMID zaznacza segment do usunięcia (będzie usunięta po odłączeniu przez wszystkich użytkowników)

pozostałe są zależne od platformy systemowej. LINX dopuszcza jeszcze IPC_INFO, SHM_INFO, SHM_STAT (jak IPC_STAT) oraz SHM_LOCK i SHM_UNLOCK (blokuje i zwalnia dostęp).

Poniższy kod źródłowy utworzy segment pamięci współdzielonej, następnie pobierze informacje o nim i zwolni pamięć.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define PROJECTID 0xef51b49e

int main( void )
{
    key_t key;
    int id,size,flag;
    struct shmid_ds buffer;
    //Inicjujemy wartości aktualne dla przyszłego wywołania shmget()
    key = ftok( "/tmp",PROJECTID );
    flag = IPC_CREAT | 0x100 | 0x80;
    size = 1024*64;
    //Tworzymy segment pamięci
    id = shmget( key,size,flag );
    //Jeżeli próba zakończona sukcesem
    if( id>0 )
    {
        //Wyświetlenie informacji diagnostycznych
        printf( "SEGMENT [key=0x%x,id=%u],process=%u\n",
        id,(unsigned)key,(unsigned)getpid() );
        //Informacje o segmencie
        (void)shmctl( id,IPC_STAT,&buffer );
        printf( "właściciel %u\n",(unsigned)(buffer.shm_cpid) );
        printf( "rozmiar %u [B]\n",(unsigned)(buffer.shm_segsz) );
        printf( "%s\n",ctime( &(buffer.shm_ctime) ) );
        //Zwalniamy i kończymy
        (void)shmctl( id,IPC_RMID,&buffer );
```

```
    }
    else{ perror( "!.!.!..shmget()..." ); exit( 1 ); }
    return 0;
}
```

Początkowo wynik to

```
$ ./shmtest
!.!.!..shmget()...: Success
```

W moim przypadku musiałem utworzyć nowy segment ponieważ nie miałem żadnych istniejących, jednak po stworzeniu i zmienieniu adresu w kodzie możemy zauważyć tym razem już poprawne wyniki

```
$ ./shmtest
SEGMENT [key=0x1,id=2656154782],process=1202
właściciel 1202
rozmiar 65536 [B]
Wed Nov 22 20:36:58 2023
```

Zanim proces będzie mógł działać na segmencie wspólnym pamięci, najpierw musi dokonać jego przyłączenia czyli odwzorowania na wskazanie adresu początkowego. Uzyskuje się to za pośrednictwem shmat() (zwolnienie następuje przez wywołanie shmdt()), efekt – w sensie operacyjnym – jest analogiczny do wywołanie funkcji przydzielających pamięć w obrębie procesu czyli malloc() czy calloc().

Przygotujemy teraz kod dwóch programów – pierwszy utworzy segment pamięci współdzielonej IPC i zainicjuje go ciągiem liczb Davida Hilberta, czyli liczb postaci

4 n + 1, gdzie n=0,1,2,3,...

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define PROJECTID 0xFF
#define n 100
int main( void )
{
    key_t key;
    int id,size,flag;
    unsigned i;
    //I oczywiście zmienna za pośrednictwem,
    //której będziemy się odwoływać do segmentu wspólnego
    unsigned *array;
    printf( "GENERATOR [%u]\n",(unsigned)getpid() );
    key = ftok( "/tmp",PROJECTID );
```

```
    flag = IPC_CREAT | 0x1B6;
    size = n*sizeof( unsigned );
    id = shmget( key,size,flag );
    //Jeżeli udało się zainicjować segment współdzielony, to
    if( id>0 )
    {
    printf( "...utworzono segment wspólny [%u][0x%x]\n\t",id,key );
    //Sposób dostępu do obszaru współdzielonego pamięci
    array = (unsigned*)shmat( id,NULL,0 );
    //...po przyłączeniu, odwołanie w sposób tradycyjny, jak do wskazania
okreslonego typu
    for( i=0;i<n;i++ )
    {
        *(array+i)=(4*i+1);
        printf( "." ); fflush( stdout);
        }
        //ponownie odłączamy segment współdzielony
        shmdt( (void*)array );
        printf( "\n...zakończono inicjację segmentu wspólnego\n" );
    }
    else{ perror( "!.!.!..shmget()..." ); exit( 1 ); }
    return 0;
}
```

Celem tego kodu było utworzenie segmentu wspólnego, więc po uruchomieniu zobaczymy taki komunikat

```
$ ./generator
GENERATOR [1430]
...utworzono segment wspólny [2][0xff51b49e]

.....................................................................................
.................
...zakończono inicjację segmentu wspólnego
```

Możemy też teraz zauważyć że nasz segment rzeczywiście znajduje się w pamięci

```
$ ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0xef51b49e 0          zciwolvo   600        65536      0
0xff51b49e 2          zciwolvo   666        400        0
```

Skoro mamy już wszystkie elementy teraz możemy napisać kod, który korzystając z naszego dzielonego segmentu obliczy pierwsze 100 elementów ciągu Davida Hilberta

```
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define PROJECTID 0xFF
#define n 100
int main( void )
{
    key_t key;
    int id,size,flag;
    struct shmid_ds buffer;
    unsigned i,*array;
    key = ftok( "/tmp",PROJECTID );
    flag = IPC_CREAT | 0x1B6;
    size = n*sizeof( unsigned );
    id = shmget( key,size,flag );
    if( id>0 )
    {
        //Poprostu przyłączmy
        array = (unsigned*)shmat( id,NULL,0 );
        //i korzystamy w zwyczajowy sposób
        printf( "100 liczb D.HILBERTa odczytanych z [%u][0x%x]\n",id,key );
        for( i=0;i<n;i++ )
        {
            printf( "%10u",*(array+i) );
            //... jeszcze tylko drobna korekta listingu
            printf( "%c",!((i+1)%5)?('\n'):(' ') );
        }
        printf( "%c",'\n' );
        //Po wykorzystaniu odłączamy
        shmdt( (void*)array );
        //i usuwamy z pamięci
        (void)shmctl( id,IPC_RMID,&buffer );
    }
    else{ perror( "!.!.!..shmget()..." ); exit( 1 ); }
    return 0;
}
```

```
$ ./hilbert
100 liczb D.HILBERTa odczytanych z [2][0xff51b49e]
         1         5         9        13        17
        21        25        29        33        37
        41        45        49        53        57
        61        65        69        73        77
        81        85        89        93        97
       101       105       109       113       117
       121       125       129       133       137
       141       145       149       153       157
       161       165       169       173       177
```

```
        181         185         189         193         197
        201         205         209         213         217
        221         225         229         233         237
        241         245         249         253         257
        261         265         269         273         277
        281         285         289         293         297
        301         305         309         313         317
        321         325         329         333         337
        341         345         349         353         357
        361         365         369         373         377
        381         385         389         393         397
```

Możemy także zweryfikować, że kod prawidłowo czyści pamięć poprzez ponowne sprawdzenie

```
$ ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0xef51b49e 0          zciwolvo   600        65536      0
```

Wnioski

1. Pamięć współdzielona w systemie UNIX: Pamięć współdzielona jest jednym z mechanizmów IPC, pozwalającym na współdzielenie danych między różnymi procesami. Dostęp do pamięci współdzielonej odbywa się poprzez tworzenie, przyłączanie, odłączanie i usuwanie segmentów pamięci. Procesy mogą dzielić dostęp do tego samego obszaru pamięci, co umożliwia efektywną komunikację i współdzielenie informacji między nimi.

2. Operacje na segmentach pamięci współdzielonej: Funkcje jak shmget(), shmctl(), shmat(), shmdt() pozwalają na tworzenie, zarządzanie oraz usuwanie segmentów pamięci współdzielonej. Dostęp do danych w pamięci współdzielonej odbywa się poprzez odwzorowanie obszaru pamięci segmentu do przestrzeni adresowej procesu. Podczas operacji zarządzania segmentem (shmctl()), można pobierać informacje o segmencie, ustawiać prawa dostępu, czy oznaczać segment do usunięcia.

3. Używanie pamięci współdzielonej: Pamięć współdzielona może być wykorzystywana do przechowywania danych między procesami. Poprzez współdzieloną pamięć możliwe jest skuteczne przekazywanie informacji, w tym przypadku sekwencji liczb, między różnymi procesami.

4. Zarządzanie i kontrola dostępu do pamięci współdzielonej: Uprawnienia dostępu do segmentów pamięci są zarządzane przez strukturę struct ipc_perm. Kluczowe jest kontrolowanie operacji na pamięci współdzielonej, takich jak przyłączanie i odłączanie segmentów, aby uniknąć wycieków pamięci lub konfliktów dostępu.