Uniwersytet Bielsko-Bialski

LABORATORIUM

Obliczeń Równoległych i Systemów Rozproszonych

Sprawozdanie nr 4

Kolejki komunikatów IPC

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

Igor Gawłowicz / 59096

Cel ćwiczenia

1. Zrozumienie Kolejek Komunikatów w IPC:
    - Poznanie roli i znaczenia kolejek komunikatów w obszarze IPC.
    - Zrozumienie ich struktury oraz sposobu interakcji z nimi.
2. Implementacja i Wykorzystanie:
    - Utworzenie kolejkę komunikatów i prezentacja sposobu zarządzania nią poprzez funkcje msgget, msgsnd, msgrcv i msgctl.
    - Przedstawienie sposobów tworzenia, wysyłania i odbierania wiadomości w kolejkach komunikatów.
3. Demonstracja Zastosowania w Praktyce:
    - Pokazanie praktycznego wykorzystania kolejek komunikatów w programach, zarówno do wysyłania, jak i odbierania komunikatów między procesami.

Przebieg ćwiczenia

Współbieżność procesów (czy wątków) wiąże się często z potrzebą okresowej (asynchronicznej) wymiany komunikatów. Żaden z omawianych wcześniej mechanizmów nie spełniłby się najlepiej w tej roli. Po raz pierwszy problem ten znalazł rozwiązanie w

UNIX System V

w postaci

kolejki komunikatów, ang. message queues

wprowadzonym jeszcze przez AT&T w roku 1983, a stosowanych do chwili obecnej w rodzinie systemów nawiązujących po spuścizny UNIX.

Kolejki komunikatów stanowią jeden z kilku elementów funkcjonalnych tworzących

InterProcess Communication, IPC

implementowany standardowo w obrębie jądra systemów będących potomkami UNIX System V.

Również **MsWindows**, którego twórcy mają od pewnego czasu aspiracje stworzenie systemu wspierającego współbieżność dysponuje także swoistym **IPC**.

Kolejka komunikatów stanowi połączona listę wiadomości przechowywanych w obszarze pamięci jądra systemowego i jest identyfikowana przez unikalny klucz – identyfikator **qmsqid (queue identifier)** – będący nieujemną liczbą całkowitą.

Z poziomu interface'u użytkownika informacja o istniejących kolejkach komunikatów może być pobrana poleceniem

```
$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

W przypadku gdyby w kolejce były opecje jakieś rekordy możemy je wyczyścić za pomocą:

```
$ ipcrm [-Q msgkey] | [-q msgid]
```

Celem obsługi tego sposobu komunikacji, w obszarze pamięci jadra tworzona jest tablica kolejek msgque[] o rozmiarze (maksymalnym) MSGMNI, której elementem składowym są struktury o definicji (w linux/msg.h)

Tablica ta jest indeksowana za pośrednictwem identyfikatora danej kolejki qid. Zauważmy, że Struktura ta zawiera w szczególności wskazania, w postaci msg_first i msg_last identyfikujące listę komunikatów powiązanych z daną kolejką.

Nowa kolejka jest tworzona, co wiąże się z "dopisaniem" do tablicy msgque struktury msqid_ds a jeżeli już istnienie, to jest otwierana celem uzyskania dostępu wywołaniem funkcji msgget().

Składowe msqid_ds do której funkcja zwróciła indeks inicjowane są w taki sposób że:

- msg_perm.cuid i msg_perm.uid oraz odpowiednio msg_perm.cgid i msg_perm.gid inicjowane są wartościami identyfikatora właściciela procesu oraz jego grupy;
- na pozycję 9 najmniej znaczących bitów pola msg_perm.mode jest kopiowanych z 9 najmniej znaczących bitów wartości aktualnej parametru formalnego msgflg, przy czym ipc_perm jest zdefiniowane w ogólności jako

```
struct ipc_perm
{
    key_t __key; /* key wysłany przez msgget() */
    uid_t uid; /* UID właściciela */
    gid_t gid; /* GID grupy właściciele */
    uid_t cuid; /* UID twórcy */
    gid_t cgid; /* GID grupy twórcy */
    unsigned short mode; /* uprawnienia */
    unsigned short seq; /* numer sekwencyjny */
};
```

- msg_qnum, msg_lspid, msg_lrpid, msg_stime i msg_rtime inicjowane są zerami (0);
- msg_ctime będzie odpowiadać aktualnemu czasowi;
- msg_qbytes przyjmuje wartość równą ograniczeniu systemowemu MSGMNB (zdefiniowanej w linux/msg.h) jako

Zauważmy, że dzięki przyjętemu sposobowi inicjowania pola msg_perm.mode za pośrednictwem argumentu flag, wykorzystując operator alternatywy bitowej '|' możemy określić w inny sposób – niż domniemany – uprawnienia flag = IPC_CREAT | mode jest to możliwe dzięki temu, że wg definicji stałej IPC_CREAT jej wartość wynosi 512, czyli binarnie 1 000 000 000 a więc właśnie 9 najmłodszych bitów jest nie wykorzystywanych.

Uprawnienia są określane, jak i w pozostałych przypadkach, za pomocą maski bitowej

```
read by user (u+r) 00400 0x100 256
write by user (u+w) 00200 0x80 128
read by group (g+r) 00040 0x20 32
write by group (g+w) 00020 0x10 16
read by others (o+r) 00004 0x4 4
write by others (o+r) 00002 0x2 2
```

Chociaż klucz key wywołaniu msgget() może być zadany w sposób dowolny, to jednak w przypadku kiedy będzie wykorzystywany już wcześniej, to funkcja zwróci błąd. W przypadku kiedy:

- ma być to kolejka prywatna albo użytkowana przez potomków utworzonych funkcją fork() (albo fork() i dalej exec*()) w zupełności wystarczającym jest podstawienie IPC_PRIVATE;
- jeżeli jednak ma kolejka ma być użytkowana przez procesy niespokrewnione, to wygodnie jest wygenerować wspólny dla wszystkich klucz wywołaniem funkcji ftok().

Ogół działań związanych z zarządzaniem kolejką, do której proces posiada uprawnienia, realizuje funkcja msgctl().

W implementacji LINUX'owej uzyskujemy tu nieco większą funkcjonalność w relacji do UNIX SysV czy POSIX.

W szczególności określając

cmd = IPC_INFO

uzyskujemy ogólne informacje odnośnie realizacji i ograniczeń kolejek komunikatów w danym systemie. Wynik zwracany jest w strukturze msginfo (definicja sys/msg.h)

```
struct msginfo
{
    int msgpool; /* --- aktualnie nieużywana --- */
    int msgmap; /* --- aktualnie nieużywana --- */
    int msgmax; /* max ilość bajtów
    jaka może być przesłana w pojedynczej wiadomości */
    int msgmnb; /* max ale ogółem do kolejki */
    int msgmni; /* max ilość kolejek */
    int msgssz; /* --- aktualnie nieużywana --- */
    int msgtql; /* --- aktualnie nieużywana --- */
    unsigned short int msgseg; /* --- aktualnie nieużywana --- */
};
```

Na podstawie tych informacji możemy napisać program, którzy utworzy nam kolejkę i przedstawi jest informację

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/msg.h>
#include <time.h>
#define PROJECTID 666
int main( void )
{
    key_t key;
    int flag, msqid;
    struct msqid_ds buffer;
    //Generujemy klucz dla potrzeb utworzenia kolejki podając pewien arbitralnie
obrany
    //(ale – koniecznie – istniejący katalog; z pewnością /tmp będzie istniał), a
ponadto pewną
    //wartość liczbową (tutaj stała PROJECTID)
    key = ftok( "/tmp", PROJECTID );
    //Następnie tworzymy kolejkę (uprawnienia odczytu i zapisu tylko dla
użytkownika)
    flag = IPC_CREAT | 0x100 | 0x80;
    msqid = msgget( key,flag );
    if( msqid<0 ){ perror( "!.!..msgget().." ); exit( 1 ); }
    else
    {
        //Jeżeli tak, to pobieramy o niej informacje
        if( !msgctl( msqid,IPC_STAT,&buffer ) )
        {
            //Identyfikator kolejki
            printf( "\tKOLEJKA [%d]\n",(int)msqid );
            //Dla pewności pobieramy nasz UID i GID
            printf( "\tuid:%d gid:%d\n",(int)getuid(),(int)getgid() );
            //... i sprawdzenia zawartości struct msqid_ds
            printf( "\t%s",ctime( &(buffer.msg_ctime ) ) );
            printf( "\tuid:%dgid:%d\n",(int)buffer.msg_perm.uid,
            (int)buffer.msg_perm.gid );
            printf( "\tkey: 0x%x\n",buffer.msg_perm.__key );
            printf( "\trozmiar kolejki: %lu B / %u MSG\n",
            buffer.msg_cbytes,(unsigned)buffer.msg_qnum );
            /* msgctl( msqid,IPC_RMID,&buffer ); */
        }
        else{ perror( "!.!..msgctl.." ); exit( 2 ); }
    }
    return 0;
}
```

Powyższy kod zwróci nam:

```
$ ./msgtest
        KOLEJKA [0]
        uid:1000 gid:1000
        Wed Nov 15 20:45:39 2023
        uid:1000gid:1000
        key: 0x9a51b49e
        rozmiar kolejki: 0 B / 0 MSG
```

Widzimy tutaj identyfikator kolejki, użytkownika, grupy do której należy, które są wzięte z systemu.

Następnie datę utworzenia tej kolejki oraz ponownie informacje o identyfikatorach w tej kolejce, tylko tym razem pobieramy je ze struktury msqid_ds, aby sprawdzić czy pokrywają się z naszymi.
Oraz ostatecznie klucz kolejki i rozmiar, jako że na razie nic nie ma w tej kolejce rozmiar jest zerowy.

Możemy teraz też ponownie sprawdzić czy nasza kolejka rzeczywiście powstała i istnieje w systemie

```
$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x9a51b49e 0          zciwolvo   600        0            0
```

Jest to dość istotna informacja ponieważ kolejki nie zamykają się samoczynnie, trzeba je usuwać manualnie więc warto o tym pamiętać.

Teraz możemy zająć się wprowadzaniem danych do kolejek, jednak najpierw przyjrzyjmy się strukturze wiadomości:

```
struct msgbuf
{
    long mtype; /* message type, must be > 0 */
    char mtext[msgsz]; /* message data */
};
```

Same wiadomości możemy wysyłać za pomocą **msgsnd()** oraz otrzymywać poprzez **msgrcv()**. W wyniku pomyślnego wykonania operacji msgsnd():

- msg_lspid będzie ustawione na ID procesu, który wysłał
- msg_qnum będzie zwiększone o 1
- msg_stime uzyska wartość bieżącej chwili czasu

natomiast msgrcv():

- zapisanie kopi komunikatu wskazywanego przez msgp do kolejki o identyfikatorze msqid oraz jego usunięcie z kolejki systemowej
- parameter msgsz określa tutaj maksymalna długość łańcucha komunikatu, jeżeli w rzeczywistości długość okaże się większa, to wynik zależy od czego czy w msgflg podano MSG_NOERROR (tak – łańcuch będzie obcięty, nie – będzie powstanie błąd, czyli zwróci -1)

W kolejnym przykładzie proces utworzy prywatną kolejką, wyśle do niej dwa komunikaty a następnie spróbuje go odebrać.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/msg.h>
#define MSGMAX 8192
#define PROJECTID 0x1A
int main( void )
{
    key_t key;
    unsigned msqid,pid;
    //Definiujemy struktury do obsługi kolejki i przesyłania komunikatów
    struct msqid_ds buffer;
    struct { long type; char text[MSGMAX]; } message;
    //I w końcu inicjujemy kolejkę komunikatów generując klucz przez ftok()
    key = ftok( "/tmp",PROJECTID );
    msqid = msgget( key,IPC_CREAT|0x100|0x80 );
    if( msqid<0 ){ perror( "!.!..msgget().." ); exit( 1 ); }
    else
    {
        //Niech komunikaty będą identyfikowanej przez PID procesu wysyłającego
        pid = getpid();
        printf( "[%u] utworzył kolejkę...%u\n",pid,msqid );
        message.type = (long)pid;
        //Przygotowujemy i wysyłamy pierwszą wiadomość
        (void) strcpy( message.text, "\'<1> pierwsza wiasomość <1>\'");
        printf( "[%u] wysyła...\t%s\n",pid,message.text );
        //Konstrukcja strlen(message.text)+1 konieczna, ponieważ msgsnd() wymaga
        //potania ilości bajtów, a strlen() zwraca długość łańcucha
        //bez końcowego '\0'
        msgsnd( msqid,(void*)&message,strlen(message.text)+1,IPC_NOWAIT );
        //... no i druga wiadomość
        (void) strcpy( message.text, "\'<2> druga wiadomość <2>\'");
        printf( "[%u] wysyła...\t%s\n",pid,message.text );
        msgsnd( msqid,(void*)&message,strlen(message.text)+1,IPC_NOWAIT );
        bzero( (void*)&message,sizeof( message ) );
        //Dopóki jeszcze coś jest w kolejce identyfikowanego typem pid, to czytamy
        while(
            msgrcv( msqid,(void*)&message,MSGMAX, pid,IPC_NOWAIT|MSG_NOERROR ) > 0
            )
```

```
        {
            printf( "[%u] otrzymał...\t%s\n",pid,message.text );
            bzero( (void*)&message,sizeof( message ) );
        }
        //Skoro nic nie pozostało, to usuwamy z pamięci naszą kolejkę
        msgctl( msqid,IPC_RMID,&buffer );
    }
    //i kończymy
    return 0;
}
```

W wyniku programu widzimy teraz

```
[1180] utworzył kolejkę...1
[1180] wysyła...        '<1> pierwsza wiasomość <1>'
[1180] wysyła...        '<2> druga wiadomość <2>'
[1180] otrzymał...      '<1> pierwsza wiasomość <1>'
[1180] otrzymał...      '<2> druga wiadomość <2>'
```

że nasza kolejka została utworzona poprawnie a nasze wiadomości wysłane prawidłowo.

Widzimy też, że prawidłowo zamknęliśmy kolejkę po pracy, więc w liście jest widoczna tylko ta pierwsza kolejka z poprzedniego zadania:

```
$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x9a51b49e 0          zciwolvo   600        0            0
```

Wnioski

1. Rola Kolejek Komunikatów w IPC:
    - Kolejki komunikatów są istotnym elementem IPC, umożliwiającym asynchroniczną wymianę komunikatów między procesami.
    - Ich struktura w systemie opartym na UNIX System V obejmuje unikalny identyfikator, uprawnienia, oraz listę przechowującą komunikaty.
2. Implementacja w Praktyce:
    - Kluczowym krokiem jest tworzenie kolejek z odpowiednimi uprawnieniami, co odbywa się poprzez funkcję msgget.
    - Wiadomości są wysyłane za pomocą msgsnd, a odbierane przez msgrcv, gdzie ważne jest określenie typu wiadomości.
    - Obsługa kolejek komunikatów odbywa się przez funkcję msgctl, umożliwiającą modyfikację i usuwanie kolejek.

3. Zastosowanie w Programach:

   - Kolejki komunikatów są używane w programach do wysyłania i odbierania danych między różnymi procesami.
   - Ich rola jest kluczowa w rozproszonych aplikacjach, gdzie wymiana danych pomiędzy procesami jest niezbędna.