Wykorzystanie semaforów i pamięci dzielonej

```
/* ============ sem.h =========== */
#define     IDE_PAM          1
#define     ROZM_PAM    100
#define     ID_SEM_ZAPISU     1
#define     ID_SEM_ODCZYTU    2

union semun {
      int val;
        struct semid_ds *buf;
        ushort *array;
};
struct bufor {
      char znak[ROZM_PAM];
};
/*============================================================================ */
      /*    sem.c */
      /* PROCES STARTOWY - uruchomienie generatora i odbiornikow */

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include "sem.h"

#define ER_SEM_ZAP      "Blad utworzenia semafora zapisu (semget) \n"
#define ER_SEM_ODCZ     "Blad utworzenia semafora odczytu (semget) \n"
#define ER_UST_ZAP      "Blad ustawienia semafora zapisu (semget) \n"
#define ER_UST_ODCZ     "Blad ustawienia semafora odczytu (semget) \n"
#define EXEC_NAD        "Blad powolania procesu nadajnika (execl)\n"
#define FORK_ODB1       "Blad powolania procesu odbiornika nr 1(fork)\n"
#define EXEC_ODB1       "Blad powolania procesu odbiornika nr 1(execl)\n"
#define FORK_ODB2       "Blad powolania procesu odbiornika nr 2(fork)\n"
#define EXEC_ODB2       "Blad powolania procesu odbiornika nr 2(execl)\n"

static int sem_odczytu,sem_zapisu;
static struct sembuf buf1[1],buf2[1]; /* czy to jest potrzebne ? */
static union semun arg;

int main()
{
  int f;
                  /* utworzenie zbiorow semaforow */
  if ((sem_zapisu=semget(ID_SEM_ZAPISU,1,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_SEM_ZAP,strlen(ER_SEM_ZAP));
      return 1; }
  if ((sem_odczytu=semget(ID_SEM_ODCZYTU,1,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_SEM_ODCZ,strlen(ER_SEM_ODCZ));
      return 1; }

                  /* ustawienie wartosci poczatkowych semaforow */
  arg.val = 1;
  if (semctl(sem_zapisu,0,SETVAL,arg) <0) {
      write(STDERR_FILENO,ER_UST_ZAP,strlen(ER_UST_ZAP));
      return 1; }
  arg.val = 0;
  if (semctl(sem_odczytu,0,SETVAL,arg) <0) {
      write(STDERR_FILENO,ER_UST_ODCZ,strlen(ER_UST_ODCZ));
      return 1; }
```

```
  if ((f=fork())==0)
  {
    execl("sem_odb","1",NULL);      /* POTOMNY: uruchomienie odbiornika nr 1 */
    write(STDERR_FILENO,EXEC_ODB1,strlen(EXEC_ODB1));
    return(1);  }
  else
    if (f == -1)
    {
      write(STDERR_FILENO,FORK_ODB1,strlen(FORK_ODB1));
      return(1); }
                        /* TO REALIZUJE TYLKO PROCES MACIERZYSTY */
    else
      if ((f=fork())==0)
      {
        execl("sem_odb","2",NULL);  /* uruchomienie procesu odbiornika nr 2 */
        write(STDERR_FILENO,EXEC_ODB2,strlen(EXEC_ODB2));
        return(1);  }
      else
        if (f == -1)
        {
          write(STDERR_FILENO,FORK_ODB2,strlen(FORK_ODB2));
          return(1); }
        else
        {
           execl("sem_nad",NULL);
           write(STDERR_FILENO,EXEC_NAD,strlen(EXEC_NAD));
           return(1); }
}
```

```
      /*    sem_nad.c   */
      /*    PROCES generatora komunikatow */
      /*    wykorzystujacy semafory i pamiec dzielona */

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include <sys/shm.h>
#include <time.h>
#include "sem.h"
#include "sem_nadf.c"

#define ER_SEM_ODCZ     "NAD:Blad kojarzenia identyfikatora semafora odczytu
(semget)\n"
#define ER_SEM_ZAP      "NAD:Blad kojarzenia identyfikatora semafora zapisu
(semget)\n"
#define ER_PAM1         "NAD:Blad definiowania pamieci dzielonej (shmget)\n"
#define ER_PAM2         "NAD:Blad okreslania adresu pamieci dzielonej (shmat)\n"
#define ER_PAM3         "NAD:Blad podczas zwalniania pamieci dzielonej (shmdt)\n"
#define ER_OP_ODCZ      "NAD:Blad operacji na semaforze odczytu (semop)\n"
#define ER_OP_ZAP "NAD:Blad operacji na semaforze zapisu (semop)\n"
#define ER_DEL_ZAP      "NAD:Blad operacji usuwania semafora zapisu (semctl)\n"
#define ER_DEL_ODCZ     "NAD:Blad operacji usuwania semafora odczytu (semctl)\n"
#define ER_DEL_PAM      "NAD:Blad operacji usuwania pamieci dzielonej(shmctl)\n"

static int sem_odczytu,sem_zapisu;
static struct sembuf buf1[1],buf2[1];
static int M, koniec;

int main()
{
  struct bufor *wiadomosc;

            /* kojarzenie identyfikatora obszaru pamieci dzielonej */
  if ((M=shmget(IDE_PAM,ROZM_PAM,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_PAM1,strlen(ER_PAM1));
      return 1; }
  if ((wiadomosc=(struct bufor *) shmat(M,0,0))==NULL) {
      write(STDERR_FILENO,ER_PAM2,strlen(ER_PAM2));
      return 1; }

            /* kojarzenie identyfikatorow zbiorow semaforow */
  if ((sem_zapisu=semget(ID_SEM_ZAPISU,1,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_SEM_ZAP,strlen(ER_SEM_ZAP));
      return 1; }
  if ((sem_odczytu=semget(ID_SEM_ODCZYTU,1,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_SEM_ODCZ,strlen(ER_SEM_ODCZ));
      return 1; }

  do
  {
    buf1[0].sem_op = -1;      /* opuszczenie semafora zapisu */
    if ((semop(sem_zapisu,buf1,1)) < 0) {
        write(STDERR_FILENO,ER_OP_ZAP,strlen(ER_OP_ZAP));
        return 1; }

    koniec=Przygotowanie_Wiadomosci(wiadomosc);
```

```
    buf2[0].sem_op=1;         /* podniesienie semafora odczytu */
    if ((semop(sem_odczytu,buf2,1)) < 0) {
        write(STDERR_FILENO,ER_OP_ODCZ,strlen(ER_OP_ODCZ));
        return 1; }
    sleep(1);
  }  while (!koniec);

  sleep(5); /* czas dla odbiornikow na zakonczenie dzialania */
            /* zwolnienie pamieci dzielonej*/
  if ((shmdt((char *) wiadomosc)) < 0) {
      write(STDERR_FILENO,ER_PAM3,strlen(ER_PAM3));
      return 1; }
  if (shmctl(M,IPC_RMID,0) <0) {
      write(STDERR_FILENO,ER_DEL_PAM,strlen(ER_DEL_PAM));
      return 1; }
  if (semctl(sem_odczytu,0,IPC_RMID,0) <0) {
      write(STDERR_FILENO,ER_DEL_ODCZ,strlen(ER_DEL_ODCZ));
      return 1; }
  if (semctl(sem_zapisu,0,IPC_RMID,0) <0) {
      write(STDERR_FILENO,ER_DEL_ZAP,strlen(ER_DEL_ZAP));
      return 1; }

  return 0;
}
```

Jeżeli pracuje więcej niż jeden odbiornik, to po wykonaniu przedstawionych funkcji, odbiorniki czekające na podniesienie semafora odczytu zasygnalizują błąd operacji na semaforze. Należy tutaj zastosować bardziej rozbudowany mechanizm kończenia pracy procesów

```
==========================================================================
      /*    sem_nadf.c  */
      /*    Funkcje uzytkowe procesu generatora komunikatow*/
      /*    wykorzystujacego semafory i pamiec dzielona */

#define DO_OPERATORA "Podaj dane (ogranicznik ENTER): "

int Przygotowanie_Wiadomosci(struct bufor *wiadomosc)
{
   char buf[ROZM_PAM];
   int dlugosc;

   write(STDOUT_FILENO,DO_OPERATORA,strlen(DO_OPERATORA));
   fgets(buf,ROZM_PAM,stdin); /* UWAGA fgets pamieta NL i doklada\0 */
   dlugosc=strlen((char *)buf)-1;
   strncpy(wiadomosc->znak,(char *)buf,dlugosc);     /* kopiowanie bez NL */
   wiadomosc->znak[dlugosc]='\0';        /* zakonczenie lancucha znakow */
   sprintf(buf,"Nadajnik przygotowal -ilosc bajtow: %d => %s\n",dlugosc,wiadomosc-
>znak);
   write(STDOUT_FILENO,buf,strlen(buf));
   return !dlugosc;
}
==========================================================================
      /*    sem_odbf.c  */
      /*    Funkcje uzytkowe procesu odbiornika komunikatow*/
      /*    wykorzystujacego semafory i pamiec dzielona */
int Odebranie_Wiadomosci(struct bufor *wiadomosc, char *nr_odbiornika)
{
   char buf[ROZM_PAM];
   int dlugosc;

   dlugosc= strlen(wiadomosc->znak);
   sprintf(buf,"Odbiornik %s odebral -ilosc bajtow: %d => %s\n",
              nr_odbiornika,dlugosc,wiadomosc->znak);
   write(STDOUT_FILENO,buf,strlen(buf));
   return !dlugosc;
}
```

```
      /*    sem_odb.c   */
      /*    PROCES odbiornika komunikatow */
      /*    wykorzystujacy semafory i pamiec dzielona */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include <sys/shm.h>
#include <time.h>
#include "sem.h"
#include "sem_odbf.c"
#define ER_SEM_ODCZ     "ODB:Blad kojarzenia identyfikatora semafora odczytu
(semget)\n"
#define ER_SEM_ZAP      "ODB:Blad kojarzenia identyfikatora semafora zapisu
(semget)\n"
#define ER_PAM1         "ODB:Blad definiowania pamieci dzielonej (shmget)\n"
#define ER_PAM2         "ODB:Blad okreslania adresu pamieci dzielonej (shmat)\n"
#define ER_PAM3         "ODB:Blad podczas zwalniania pamieci dzielonej (shmdt)\n"
#define ER_OP_ODCZ      "ODB:Blad operacji na semaforze odczytu (semop)\n"
#define ER_OP_ZAP "ODB:Blad operacji na semaforze zapisu (semop)\n"
static int sem_odczytu,sem_zapisu;
static struct sembuf buf1[1],buf2[1];
static int M;
int main(int arg, char **arc)
{
  struct bufor *wiadomosc;
  int koniec;
            /* kojarzenie identyfikatora obszaru pamieci dzielonej */
  if ((M=shmget(IDE_PAM,ROZM_PAM,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_PAM1,strlen(ER_PAM1));
      return 1; }
  if ((wiadomosc=(struct bufor *) shmat(M,0,0))==NULL) {
      write(STDERR_FILENO,ER_PAM2,strlen(ER_PAM2));
      return 1; }
            /* kojarzenie identyfikatorow zbiorow semaforow */
  if ((sem_zapisu=semget(ID_SEM_ZAPISU,1,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_SEM_ZAP,strlen(ER_SEM_ZAP));
      return 1; }
  if ((sem_odczytu=semget(ID_SEM_ODCZYTU,1,IPC_CREAT | 0666)) < 0) {
      write(STDERR_FILENO,ER_SEM_ODCZ,strlen(ER_SEM_ODCZ));
      return 1; }
 do
  {
    buf2[0].sem_op = -1; /* opuszczenie semafora odczytu */
    if ((semop(sem_odczytu,buf2,1)) < 0) {
        write(STDERR_FILENO,ER_OP_ODCZ,strlen(ER_OP_ODCZ));
        return 1; }
    koniec=Odebranie_Wiadomosci(wiadomosc, arc[0]);
    buf1[0].sem_op=1;   /* podniesienie semafora zapisu */
    if ((semop(sem_zapisu,buf1,1)) < 0) {
        write(STDERR_FILENO,ER_OP_ZAP,strlen(ER_OP_ZAP));
        return 1; }
   } while (!koniec);

  if ((shmdt((char *) wiadomosc)) < 0) {
      write(STDERR_FILENO,ER_PAM3,strlen(ER_PAM3));
      return 1; }
  return 0;
}
```

/* ============== kom.h ===========*/

```
/* przyklad wykorzystania kolejek komunikatow */

#define     NR_KANALU_1 1
#define     NR_KANALU_2 2
#define     ROZM_KANALU 256

struct komunikat {
      long typ;
      char znak[ROZM_KANALU];
};
```

```
================================================================================
```

```
      /*    kom.c */
      /* PROCES STARTOWY - uruchomienie procesow klienta i serwera
                  wykorzystujacych kolejki komunikatow */

#include <stdio.h>
#include <unistd.h>

#define EXEC_SW "Blad powolania procesu serwera (execl)\n"
#define FORK_SW "Blad powolania procesu serwera (fork)\n"
#define EXEC_KL "Blad powolania procesu klienta (execl)\n"

int main()
{
  int f;

  if ((f=fork())==0)
  {
    execl("kom_sw",NULL);     /* POTOMNY: uruchomienie serwera */
    write(STDERR_FILENO,EXEC_SW,strlen(EXEC_SW));
    return(1);  }
  else
    if (f == -1)
    {
      write(STDERR_FILENO,FORK_SW,strlen(FORK_SW));
      return(1); }
    else
    {    execl("kom_kl",NULL);      /* MACIERZYSTY: uruchomienie klienta */
         write(STDERR_FILENO,EXEC_KL,strlen(EXEC_KL));
         return(1); }
}
```

```
      /*    kom_serw.c  */
      /* Proces SERWERA wykorzystujacego kolejki komunikatow */
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "kom.h"
#include "kom_swf.c"
#define KOM_ODB "SW:Blad odbierania od klienta (msgrcv)\n"
#define KOM_NAD "SW:Blad nadawania do klienta (msgsnd)\n"
#define KOM_KOL1 "SW:Blad tworzenia kolejki od klienta do serwera(msgget)\n"
#define KOM_KOL2 "SW:Blad tworzenia kolejki od serwera do klienta(msgget)\n"
static int Kanal_WY, Kanal_WE;
static struct komunikat kom;

int main()
{
  struct msqid_ds bufp_WY, bufp_WE;
  int dlugosc, koniec;
                    /* definiowanie kolejki od klienta do serwera*/
  if ((Kanal_WE = msgget(NR_KANALU_1,IPC_CREAT|0666)) < 0) {
      write(STDERR_FILENO,KOM_KOL1,strlen(KOM_KOL1));
      return 1; }
                    /* definiowanie kolejki od serwera do klienta*/
  if ((Kanal_WY = msgget(NR_KANALU_2,IPC_CREAT|0666)) < 0) {
      write(STDERR_FILENO,KOM_KOL2,strlen(KOM_KOL2));
      return 1; }

  do
  {
    if ((dlugosc=msgrcv(Kanal_WE, &kom, ROZM_KANALU, 0, 0)) < 0) {
       write(STDERR_FILENO, KOM_ODB, strlen(KOM_ODB));
       return 1; };
    koniec=Obsluga_Zgloszenia(&kom,dlugosc);
    if (msgsnd(Kanal_WY, &kom, dlugosc, 0) < 0) {
       write(STDERR_FILENO, KOM_NAD, strlen(KOM_NAD));
       return 1;};
     }
  while(!koniec);
}
/* ================================================================ */
      /*    kom_swf.c   */
      /* Funkcje przetwarzania dla procesu SERWERA  */
        /* wykorzystujacego kolejki komunikatow */

int Obsluga_Zgloszenia(struct komunikat *kom, int dlugosc)
{
   char xx[256];
   kom->znak[dlugosc]='\0';         /* zakonczenie lancucha znakow */
   sprintf(xx,"Serwer odebral -ilosc bajtow: %d => %X %s\n",
         dlugosc,kom->typ,kom->znak);
   write(STDOUT_FILENO,xx,strlen(xx));
   if (dlugosc) (kom->typ)+=dlugosc;
      else kom->typ=1;
   sprintf(xx,"Serwer przygotowal -ilosc bajtow: %d => %X %s\n",
              dlugosc,kom->typ,kom->znak);
   write(STDOUT_FILENO,xx,strlen(xx));
   return !dlugosc;
}
```

```
      /*    kom_klient.c      */
      /* Proces KLIENTA wykorzystujacego kolejki komunikatow */

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "kom.h"
#include "kom_klf.c"

#define KOM_ODB "KL:Blad odbierania od serwera (msgrcv)\n"
#define KOM_NAD "KL:Blad nadawania do serwera (msgsnd)\n"
#define KOM_KOL1 "KL:Blad tworzenia kolejki od klienta do serwera(msgget)\n"
#define KOM_KOL2 "KL:Blad tworzenia kolejki od serwera do klienta(msgget)\n"
#define KOM_DEL1 "KL:Blad usuwania kolejki od klienta do serwera(RMID)\n"
#define KOM_DEL2 "KL:Blad usuwania kolejki od serwera do klienta(RMID)\n"

static int Kanal_WY, Kanal_WE;
static struct komunikat kom;

int main()
{
  struct msqid_ds bufp_WY, bufp_WE;
  int dlugosc, koniec;
                    /* definiowanie kolejki od klienta do serwera*/
  if ((Kanal_WY = msgget(NR_KANALU_1,IPC_CREAT|0666)) < 0) {
      write(STDERR_FILENO,KOM_KOL1,strlen(KOM_KOL1));
      return 1; }
                    /* definiowanie kolejki od serwera do klienta*/
  if ((Kanal_WE = msgget(NR_KANALU_2,IPC_CREAT|0666)) < 0) {
      write(STDERR_FILENO,KOM_KOL2,strlen(KOM_KOL2));
      return 1; }
do
  {
    Przygotowanie_Komunikatu(&kom, &dlugosc);
    if (msgsnd(Kanal_WY, &kom, dlugosc, 0) < 0) {
       write(STDERR_FILENO,KOM_NAD,strlen(KOM_NAD));
       return 1; }
    if ((dlugosc=msgrcv(Kanal_WE, &kom, ROZM_KANALU, 0, 0)) < 0) {
       write(STDERR_FILENO,KOM_ODB,strlen(KOM_ODB));
       return 1; };
    koniec=Obrobka_Odpowiedzi_Serwera(&kom, dlugosc);
  } while( !koniec);

  if (msgctl(Kanal_WY, IPC_RMID, &bufp_WY) < 0)
     write(STDERR_FILENO,KOM_KOL1,strlen(KOM_KOL1));
  if (msgctl(Kanal_WE, IPC_RMID, &bufp_WE) < 0)
      write(STDERR_FILENO,KOM_KOL2,strlen(KOM_KOL2));

}
```

```
      /*    kom_klf.c   */
      /* Funkcje przetwarzania dla procesu KLIENTA  */
         /* wykorzystujacego kolejki komunikatow */

#define DO_OPERATORA "Podaj dane (ogranicznik ENTER): "

/* ============================================================ */

int Przygotowanie_Komunikatu(struct komunikat *kom, int *dlugosc)
{
   char buf[256];
   write(STDOUT_FILENO,DO_OPERATORA,strlen(DO_OPERATORA));
   fgets(buf,256,stdin);      /* UWAGA fgets pamieta NL i doklada\0 */
   *dlugosc=strlen((char *)buf)-1;
   strncpy(kom->znak,(char *)buf,*dlugosc);    /* kopiowanie bez NL */
   kom->znak[*dlugosc]='\0';        /* zakonczenie lancucha znakow */
   kom->typ=5;
   sprintf(buf,"Klient przygotowal -ilosc bajtow: %d => %X %s\n",
              *dlugosc,kom->typ,kom->znak);
   write(STDOUT_FILENO,buf,strlen(buf));
   return 0;
}

/* ============================================================ */

int Obrobka_Odpowiedzi_Serwera(struct komunikat *kom, int dlugosc)
{
   char xx[256];

   sprintf(xx,"Klient odebral -ilosc bajtow: %d => %X %s\n",
              dlugosc,kom->typ,kom->znak);
   write(STDOUT_FILENO,xx,strlen(xx));
   return !dlugosc;
}
```

Łącza nienazwane - przykład A

```
      /*    pipe.h      */
      /* Plik naglowkowy dla procesow  */
      /* wykorzystujacych lacza nienazwane (pipe's) */
#define     OGRANICZNIK 'z'
#define     LEN         100
/* ========================================================================= */
      /*    pipe.c      */
      /* Komunikacja dwukierunkowa procesow przez lacze nienazwane (pipe)*/
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "pipe.h"
#define KOM_PIPE1 "Blad utworzenia lacza DO_POTOMKA (pipe)\n"
#define KOM_PIPE2 "Blad utworzenia lacza DO_RODZICA (pipe)\n"
#define FORK      "Blad powolania procesu potomnego (fork)\n"
#define EXEC_SW   "Blad powolania procesu serwera (execl)\n"
#define EXEC_KL   "Blad powolania procesu klienta (execl)\n"
int do_potomka[2], do_rodzica[2];
int main()
{
  int f;
  char param1[10], param2[10];

  if (pipe(do_potomka) < 0) {
      write(STDERR_FILENO,KOM_PIPE1,strlen(KOM_PIPE1));
      return 1; }
/*    Po wykonaniu powyzszej funkcji pipe w tablicy deskryptorów plikow pojawia sie
      dwie nowe pozycje (3 i 4)
*/
  if (pipe(do_rodzica) < 0) {
      write(STDERR_FILENO,KOM_PIPE2,strlen(KOM_PIPE2));
      return 1; }
/*    Po wykonaniu powyzszej funkcji pipe w tablicy deskryptorów plikow pojawia sie
      dwie nowe pozycje (5 i 6)
*/
  if ((f=fork()) == -1)
     {
      write(STDERR_FILENO,FORK,strlen(FORK));
      return(1); }
  else
    if (f == 0)
    {                   /* POTOMNY: to jest kod procesu potomnego - SERWERa*/
      close(do_potomka[1]);   /* zamkniecie konca lacza przeznaczonego do pisania */
      close(do_rodzica[0]);   /* zamkniecie konca lacza przeznaczonego do czytania */
      sprintf(param1,"%d",do_potomka[0]); /* lub itoa(do_potomka[0],param1,10) */
      sprintf(param2,"%d",do_rodzica[1]); /* lub itoa(do_rodzica[1],param2,10); */
      execl("pipe_sw",param1,param2,NULL);
      write(STDERR_FILENO,EXEC_SW,strlen(EXEC_SW));
      return(1); }
    else
    {                   /* MACIERZYSTY: to jest kod procesu macierzystego - KLIENTa
*/
      close(do_potomka[0]);   /* zamkniecie konca lacza przeznaczonego do czytania */
      close(do_rodzica[1]);   /* zamkniecie konca lacza przeznaczonego do pisania */
      sprintf(param1,"%d",do_rodzica[0]); /* lub itoa(do_rodzica[0],param1,10); */
      sprintf(param2,"%d",do_potomka[1]); /* lub itoa(do_potomka[1],param2,10); */
      execl("pipe_kl",param1,param2,NULL);
      write(STDERR_FILENO,EXEC_KL,strlen(EXEC_KL));
      return(1);
    }}
```

Łącza nienazwane - przykład A- cd.

```
      /*    pipe_kl.c   */
      /* Proces KLIENTA
            wykorzystujacy do komunikacji lacze nienazwane (pipe's) */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "pipe.h"
#include "pipe_klf.c"

#define KLI_NAD         "Blad pisania po stronie procesu klienta\n"
#define KLI_ODB         "Blad czytania po stronie procesu klienta\n"

char buf_we[LEN], buf_wy[LEN];

int main(int arg, char **arc)
{
   int koniec,dlugosc;
   int Kanal_WE, Kanal_WY;
   char xx[50];

   Kanal_WE=atoi(arc[0]);
   Kanal_WY=atoi(arc[1]);
   sprintf(xx,"KLIENT: WE=%d  WY=%d\n",Kanal_WE,Kanal_WY);
   write(STDOUT_FILENO,xx,strlen(xx));
   sleep(2);
   do
   {
      Przygotowanie_Wiadomosci(buf_wy,&dlugosc);
      if (write(Kanal_WY,buf_wy,dlugosc) < 0) {
          write(STDERR_FILENO,KLI_NAD,strlen(KLI_NAD));
          return 1; }
      if ((dlugosc=read(Kanal_WE,&buf_we,LEN)) < 0) {
          write(STDERR_FILENO,KLI_ODB,strlen(KLI_ODB));
          return 1; };
      koniec=Obrobka_Odpowiedzi_Serwera(buf_we, dlugosc);
   } while (!koniec);
}
```

Łącza nienazwane - przykład A- cd.

```
      /*    pipe_sw.c   */
      /* Proces SERWERA
            wykorzystujacy do komunikacji lacze nienazwane (pipe's) */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "pipe.h"
#include "pipe_swf.c"

#define SRV_NAD         "Blad pisania po stronie procesu serwera\n"
#define SRV_ODB         "Blad czytania po stronie procesu serwera\n"

char buf[LEN];

int main(int arg, char **arc)
{
   int koniec,dlugosc;
   int Kanal_WE, Kanal_WY;
   char xx[50];

   Kanal_WE=atoi(arc[0]);
   Kanal_WY=atoi(arc[1]);
   sprintf(xx,"SERWER: WE=%d  WY=%d\n",Kanal_WE,Kanal_WY);
   write(STDOUT_FILENO,xx,strlen(xx));
   sleep(2);
   do
   {
      if ((dlugosc=read(Kanal_WE,&buf,LEN)) < 0) {
          write(STDERR_FILENO,SRV_ODB,strlen(SRV_ODB));
          return 1; };
      koniec=Obsluga_Zgloszenia(buf,&dlugosc);
      if (write(Kanal_WY,buf,dlugosc) < 0) {
          write(STDERR_FILENO,SRV_NAD,strlen(SRV_NAD));
          return 1; }
   } while (!koniec);
}
```

Łącza nienazwane - przykład A- cd.

```
      /*    pipe_klf.c  */
      /* Funkcje przetwarzania dla procesu KLIENTA  */
      /* wykorzystujacego lacza nienazwane (pipe's) */

#define DO_OPERATORA "Podaj dane (ogranicznik 'z'): "

/* ============================================================ */
int Przygotowanie_Wiadomosci(char kom[], int *dlugosc)
{
   char buf[256];
   write(STDOUT_FILENO,DO_OPERATORA,strlen(DO_OPERATORA));
   fgets(buf,256,stdin);           /* UWAGA fgets pamieta NL i doklada\0 */
   *dlugosc=strlen((char *)buf)-1;
   strncpy((char *)kom,(char *)buf,*dlugosc);  /* kopiowanie bez NL */
   kom[*dlugosc]='\0';             /* zakonczenie lancucha znakow */
   sprintf(buf,"Klient przygotowal -ilosc bajtow: %d => %s\n",*dlugosc,kom);
   write(STDOUT_FILENO,buf,strlen(buf));
   return 0;
}
/* ============================================================ */
int Obrobka_Odpowiedzi_Serwera(char kom[], int dlugosc)
{
   char xx[256];

   sprintf(xx,"Klient odebral -ilosc bajtow: %d => %s\n",dlugosc,kom);
   write(STDOUT_FILENO,xx,strlen(xx));
   if (kom[0]==OGRANICZNIK) return 1;
   else return 0;
}
```

```
      /*    pipe_swf.c  */
      /* Funkcje przetwarzania dla procesu SERWERA  */
        /* wykorzystujacego lacza nienazwane (pipe's) */
/* ============================================================ */
#define  SUFIX    "<<<+SRV"

int Obsluga_Zgloszenia(char kom[], int *dlugosc)
{
   char xx[256];
   kom[*dlugosc]='\0';        /* zakonczenie lancucha znakow */
   sprintf(xx,"Serwer odebral -ilosc bajtow: %d => %s\n",*dlugosc,kom);
   write(STDOUT_FILENO,xx,strlen(xx));
   strcat((char *)kom,SUFIX);
   *dlugosc=strlen((char *)kom);
   sprintf(xx,"Serwer przygotowal -ilosc bajtow: %d => %s\n",
              *dlugosc,kom);
   write(STDOUT_FILENO,xx,strlen(xx));
   if (kom[0]==OGRANICZNIK) return 1;
   else return 0;
}
```

Łącza nienazwane - przykład C

```
      /*    pipe.h      */
      /* Plik naglowkowy dla procesow  */
      /* wykorzystujacych lacza nienazwane (pipe's) */
/* ============================================================ */
#define     OGRANICZNIK 'z'
#define     LEN         100
```

```
      /*    pipe2.c     */
      /* Komunikacja dwukierunkowa procesow przez lacze nienazwane (pipe's)*/
      /* Wykorzystanie funkcji dup i deskryptorow plikow stdin i stdout */

#include <stdio.h>
#include <unistd.h>

#define KOM_PIPE1 "Blad utworzenia lacza DO_POTOMKA (pipe)\n"
#define KOM_PIPE2 "Blad utworzenia lacza DO_RODZICA (pipe)\n"
#define FORK            "Blad powolania procesu potomnego (fork)\n"
#define EXEC_SW         "Blad powolania procesu serwera (execl)\n"
#define EXEC_KL         "Blad powolania procesu klienta (execl)\n"
#define KOM_DUP1  "Blad skopiowania konca lacza do czytania w procesie serwera
(dup)\n"
#define KOM_DUP2  "Blad skopiowania konca lacza do pisania w procesie serwera
(dup)\n"
#define KOM_DUP3  "Blad skopiowania konca lacza do pisania w procesie klienta
(dup)\n"
#define KOM_DUP4  "Blad skopiowania konca lacza do czytania w procesie klienta
(dup)\n"

int do_potomka[2], do_rodzica[2];

int main()
{
  int f,koniec,dlugosc;

  if (pipe(do_potomka) < 0) {
      write(STDERR_FILENO,KOM_PIPE1,strlen(KOM_PIPE1));
      return 1; }

  if (pipe(do_rodzica) < 0) {
      write(STDERR_FILENO,KOM_PIPE2,strlen(KOM_PIPE2));
      return 1; }

  if ((f=fork()) == -1)
     {
      write(STDERR_FILENO,FORK,strlen(FORK));
      return(1); }
```

```
else
    if (f == 0)
    {                   /* POTOMNY: to jest kod procesu potomnego - SERWERa*/
      close(0);
      if ((dup(do_potomka[0])) < 0) {     /* zdublowanie konca lacza  do czytania */
        write(STDERR_FILENO,KOM_DUP1,strlen(KOM_DUP1));  /* jako standardowego
wejscia */
          return(1); }

      close(1);
      if ((dup(do_rodzica[1])) < 0) {     /* zdublowanie konca lacza  do pisania */
          write(STDERR_FILENO,KOM_DUP2,strlen(KOM_DUP2));/* jako standardowego
wyjscia */
          return(1); }

      close(do_rodzica[1]);   /* zamkniecie zbednych deskryptorow */
      close(do_potomka[0]);
      close(do_rodzica[0]);
      close(do_potomka[1]);

      execl("pipe_sw2",NULL);
      write(STDERR_FILENO,EXEC_SW,strlen(EXEC_SW));
      return(1);
    }
    else
    {                   /* MACIERZYSTY: to jest kod procesu macierzystego - KLIENTa
*/
      close(1);
      if ((dup(do_potomka[1])) < 0) {     /* zdublowanie konca lacza do pisania */
          write(STDERR_FILENO,KOM_DUP3,strlen(KOM_DUP3));/* jako standardowego
wyjscia */
          return(1); }

      close(0);
      if ((dup(do_rodzica[0])) < 0) {     /* zdublowanie konca lacza do czytania */
          write(STDERR_FILENO,KOM_DUP4,strlen(KOM_DUP4));/* jako standardowego
wejscia */
          return(1); }

      close(do_potomka[1]);   /* zamkniecie zbednych deskryptorow */
      close(do_rodzica[0]);
      close(do_potomka[0]);
      close(do_rodzica[1]);

      execl("pipe_kl2",NULL);
      write(STDERR_FILENO,EXEC_KL,strlen(EXEC_KL));
      return(1);
    }
}
```

Łącza nienazwane - przykład C cd.

```
      /*    pipe_kl2.c  */
      /* Proces KIENTA wykorzystujacy do komunikacji lacze nienazwane (pipe's) */
      /* Wykorzystanie funkcji dup i deskryptorow plikow stdin i stdout */
#include <stdio.h>
#include <unistd.h>
#define     LEN 100
#define KLI_NAD         "Blad pisania po stronie procesu klienta\n"
#define KLI_ODB         "Blad czytania po stronie procesu klienta\n"
char xx[LEN],buf_we[LEN],buf_wy[LEN];
int main()
{
   int i;
   for (i=1; i<5; i++)
   {  sprintf(buf_wy,"Komunikat %d",i);
      sprintf(xx,"KLIENT przygotowal: %s\n",buf_wy);
      write(STDERR_FILENO,xx,strlen(xx));
      if (write(STDOUT_FILENO,buf_wy,strlen(buf_wy)) < 0) {
          write(STDERR_FILENO,KLI_NAD,strlen(KLI_NAD));
          return 1; }
      if ((read(STDIN_FILENO,buf_we,LEN)) < 0) {
          write(STDERR_FILENO,KLI_ODB,strlen(KLI_ODB));
          return 1; };
      sprintf(xx,"KLIENT odebral: %s\n\n",buf_we);
      write(STDERR_FILENO,xx,strlen(xx));
      sleep(3);
   }
}
```

```
      /*    pipe_sw2.c  */
      /* Proces SERWERA wykorzystujacy do komunikacji lacze nienazwane (pipe's) */
      /* Wykorzystanie funkcji dup i deskryptorow plikow stdin i stdout */

#include <stdio.h>
#include <unistd.h>
#define LEN 100
#define SRV_NAD         "Blad pisania po stronie procesu serwera\n"
#define SRV_ODB         "Blad czytania po stronie procesu serwera\n"
char xx[LEN],buf_we[LEN],buf_wy[LEN];

int main()
{
   int k,dlugosc;
   k=1;
   do
   {
      if ((dlugosc=read(STDIN_FILENO,buf_we,LEN)) < 0) {
          write(STDERR_FILENO,SRV_ODB,strlen(SRV_ODB));
          return 1; };
      if (dlugosc==0) return 0;     /* ten warunek zostanie spelniony gdy nie bedzie
                                       juz procesow piszacych do lacza */
      sprintf(xx,"SERWER odebral: %s\n",buf_we);
      write(STDERR_FILENO,xx,strlen(xx));
      sprintf(buf_wy,"SRV%d: %s",k++,buf_we);
      sprintf(xx,"SERWER przygotowal: %s\n",buf_wy);
      write(STDERR_FILENO,xx,strlen(xx));
      if (write(STDOUT_FILENO,buf_wy,strlen(buf_wy)) < 0) {
          write(STDERR_FILENO,SRV_NAD,strlen(SRV_NAD));
          return 1; }
   } while (1);
}
```

```
      /*    pipe.h      */
      /* Plik naglowkowy dla procesow wykorzystujacych lacza nazwane */
      /* kolejki FIFO */

#define     FIFO_DO_KLIENTA         "./fifo_do_kl"
#define     FIFO_DO_SERWERA         "./fifo_do_sw"
#define     OGRANICZNIK 'z'
#define     LEN         100
/* ============================================================ */
      /*    fifo.c      */
      /* Komunikacja dwukierunkowa procesow przez lacze nazwane */
      /* kolejka FIFO */
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include "fifo.h"
#define UNLINK_KL       "Blad usuwania kolejki FIFO do klienta (unlink)\n"
#define UNLINK_SW       "Blad usuwania kolejki FIFO do serwera (unlink)\n"
#define KOM_FIFO_KL     "Blad utworzenia kolejki FIFO do klienta (mknod)\n"
#define KOM_FIFO_SW     "Blad utworzenia kolejki FIFO do serwera (mknod)\n"
#define FORK            "Blad powolania procesu potomnego (fork)\n"
#define EXEC_SW         "Blad powolania procesu serwera (execl)\n"
#define EXEC_KL         "Blad powolania procesu klienta (execl)\n"

int main()
{
  int f;

  if (access(FIFO_DO_KLIENTA,F_OK) != -1) {
  if (unlink(FIFO_DO_KLIENTA) < 0) {
      write(STDERR_FILENO,UNLINK_KL,strlen(UNLINK_KL));
      return 1; }}
  if (access(FIFO_DO_SERWERA,F_OK) != -1) {
  if (unlink(FIFO_DO_SERWERA) < 0) {
      write(STDERR_FILENO,UNLINK_SW,strlen(UNLINK_SW));
      return 1; }}
  if (mknod(FIFO_DO_KLIENTA,S_IFIFO|0666,0) < 0) {
      write(STDERR_FILENO,KOM_FIFO_KL,strlen(KOM_FIFO_KL));
      return 1; }
  if (mknod(FIFO_DO_SERWERA,S_IFIFO|0666,0) < 0) {
      write(STDERR_FILENO,KOM_FIFO_SW,strlen(KOM_FIFO_SW));
      return 1; }

  if ((f=fork()) == -1)
     {
      write(STDERR_FILENO,FORK,strlen(FORK));
      return(1); }
  else
    if (f == 0)
    {                   /* POTOMNY: to jest kod procesu potomnego - SERWERa*/
      execl("fifo_sw",NULL);
      write(STDERR_FILENO,EXEC_SW,strlen(EXEC_SW));
      return(1);
    }
    else
    {             /* MACIERZYSTY: to jest kod procesu macierzystego - KLIENTa */
      execl("fifo_kl",NULL);
      write(STDERR_FILENO,EXEC_KL,strlen(EXEC_KL));
      return(1);
    }
```

```
}

      /*    fifo_kl.c   */
      /* Proces KIENTA wykorzystujacy do komunikacji lacze nazwane */
      /* kolejki FIFO */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include "fifo.h"
#include "fifo_klf.c"

#define KLI_NAD         "Blad pisania po stronie procesu klienta\n"
#define KLI_ODB         "Blad czytania po stronie procesu klienta\n"
#define OPEN_SW         "KLIENT: Blad otwarcia kolejki FIFO do serwera\n"
#define OPEN_KL         "KLIENT: Blad otwarcia kolejki FIFO do klienta\n"

char buf_we[LEN], buf_wy[LEN];

int main()
{
   int koniec,dlugosc;
   int Kanal_WE, Kanal_WY;

   if ((Kanal_WY=open(FIFO_DO_SERWERA,O_WRONLY)) < 0) {
      write(STDERR_FILENO,OPEN_SW,strlen(OPEN_SW));
      return 1; }
   if ((Kanal_WE=open(FIFO_DO_KLIENTA,O_RDONLY)) < 0) {
      write(STDERR_FILENO,OPEN_KL,strlen(OPEN_KL));
      return 1; }

   do
   {
      Przygotowanie_Wiadomosci(buf_wy,&dlugosc);
      if (write(Kanal_WY,buf_wy,dlugosc) < 0) {
          write(STDERR_FILENO,KLI_NAD,strlen(KLI_NAD));
          return 1; }
      if ((dlugosc=read(Kanal_WE,&buf_we,LEN)) < 0) {
          write(STDERR_FILENO,KLI_ODB,strlen(KLI_ODB));
          return 1; };
      koniec=Obrobka_Odpowiedzi_Serwera(buf_we, dlugosc);
   } while (!koniec);
}
```

```
      /*    fifo_sw.c   */
      /* Proces KIENTA wykorzystujacy do komunikacji lacze nazwane */
      /* kolejki FIFO */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include "fifo.h"
#include "fifo_swf.c"

#define SRV_NAD         "Blad pisania po stronie procesu serwera\n"
#define SRV_ODB         "Blad czytania po stronie procesu serwera\n"
#define OPEN_SW         "SERWER: Blad otwarcia kolejki FIFO do serwera\n"
#define OPEN_KL         "SERWER: Blad otwarcia kolejki FIFO do klienta\n"


char buf[LEN];

int main()
{
   int koniec,dlugosc;
   int Kanal_WE, Kanal_WY;

   if ((Kanal_WE=open(FIFO_DO_SERWERA,O_RDONLY)) < 0) {
      write(STDERR_FILENO,OPEN_SW,strlen(OPEN_SW));
      return 1; }
   if ((Kanal_WY=open(FIFO_DO_KLIENTA,O_WRONLY)) < 0) {
      write(STDERR_FILENO,OPEN_KL,strlen(OPEN_KL));
      return 1; }

   do
   {
      if ((dlugosc=read(Kanal_WE,&buf,LEN)) < 0) {
          write(STDERR_FILENO,SRV_ODB,strlen(SRV_ODB));
          return 1; };
      koniec=Obsluga_Zgloszenia(buf,&dlugosc);
      if (write(Kanal_WY,buf,dlugosc) < 0) {
          write(STDERR_FILENO,SRV_NAD,strlen(SRV_NAD));
          return 1; }
   } while (!koniec);
}
```