4. Komunikacja pomiędzy procesami przez łącza nienazwane i nazwane

Łącza nienazwane (*ang. Unnamed Pipes*) i nazwane (*ang. Named Pipes*) - jedna z historycznie pierwszych metod komunikacji międzyprocesowej. Wywodzą się z systemu UNIX. Do implementacji komunikacji międzyprocesowej wykorzystują znaną koncepcję plików.

4.1 Niskopoziomowe funkcje dostępu do plików

Niskopoziomowe funkcje dostępu do plików zapewniają dostęp do plików regularnych a także specjalnych jak:

- łącza nazwane
- nienazwane
- gniazdka
- urządzenia.

Nr	Funkcja	Opis
1	**open**	Otwarcie lub utworzenie pliku
2	**read**	Odczyt z pliku
3	**write**	Zapis do pliku
4	**lseek**	Pozycjonowanie bieżącej pozycji pliku
5	**fcntl**	Ustawianie i testowanie różnorodnych atrybutów pliku
6	**close**	Zamknięcie pliku

Tab. 4-1 Ważniejsze niskopoziomowe funkcje dostępu do plików

Otwarcie pliku – funkcja open

```
int open(char *path,int oflag,…)
```

path Nazwa pliku lub urządzenia
oflag Tryb dostępu do pliku – składa się z bitów – opis w pliku nagłówkowym <fcntl.h>

Funkcja powoduje otwarcie pliku lub urządzenia o nazwie wyspecyfikowanej w parametrze path. Otwarcie następuje zgodnie z trybem oflag.

PDF created with pdfFactory trial version www.pdffactory.com

Funkcja zwraca:
 > 0 – uchwyt do pliku (*ang. File handle*) – mała liczba typu int.
- 1 – gdy wystąpił błąd.

O_RDONLY	Tylko odczyt
O_WRONLY	Tylko zapis
O_RDWR	Odczyt i zapis
O_CREAT	Utwórz plik gdy nie istnieje
O_NONBLOCK	Gdy flaga ustawiona odczyt i zapis mogą być operacjami blokującymi

Tab. 4-2 Ważniejsze tryby dostępu – parametr oflag

Odczyt z pliku – funkcja read

```
int read(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są przeczytane bajty
nbytes Liczba bajtów którą chcemy przeczytać.

Funkcja powoduje odczyt z pliku identyfikowanego przez fdes, nbytes bajtów i umieszczenie ich w buforze.
Funkcja zwraca:
 > 0 – liczbę rzeczywiście przeczytanych bajtów,
- 1 – gdy błąd.

Zapis do pliku – funkcja write

```
int write(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są bajty przeznaczone do zapisu
nbytes Liczba bajtów którą chcemy zapisać

Funkcja powoduje zapis do pliku identyfikowanego przez fdes nbytes bajtów znajdujących buforze.
Funkcja zwraca:
 > 0 – liczbę rzeczywiście zapisanych bajtów,
- 1 – gdy błąd.

PDF created with pdfFactory trial version www.pdffactory.com

Zamknięcie pliku – funkcja close

`int close(int fdes)`

`fdes` Uchwyt do pliku zwracany przez funkcję open

Funkcja powoduje zamknięcie pliku identyfikowanego przez fdes. Należy ją wykonać gdy nie będą już wykonywane operacje na danym pliku .

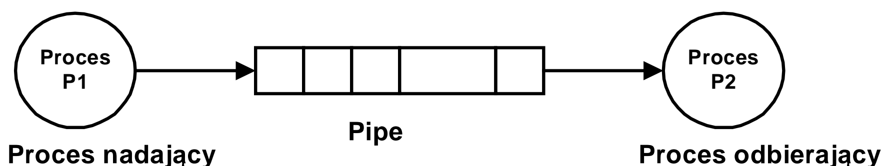
```
main(void) {
  int fd;
  char buf[80];
  fd = open(„/home/jan/mój_plik.txt”,O_RDONLY);
  do {
     rd = read(fd,buf,80);
     printf("Odczytano %d bajtów\n”,rd);
  } while(rd == 80);
  close(fd);
}
```

Przykład 4-1 **Przykład odczytu pliku tekstowego**

PDF created with pdfFactory trial version www.pdffactory.com

4.2 Łącza nienazwane

Łącze nienazwane *(ang. Pipe*) można wyobrazić sobie jako rodzaj „rury bitowej” łączącej dwa procesy. Łącze nienazwane implementowane jest jako bufor cykliczny.



Rys. 1 Procesy P1 i P2 komunikują się poprzez łącze nienazwane

Łącze tworzy się poprzez wykonanie funkcji pipe:

```
int pipe(int fildes[2]);
```

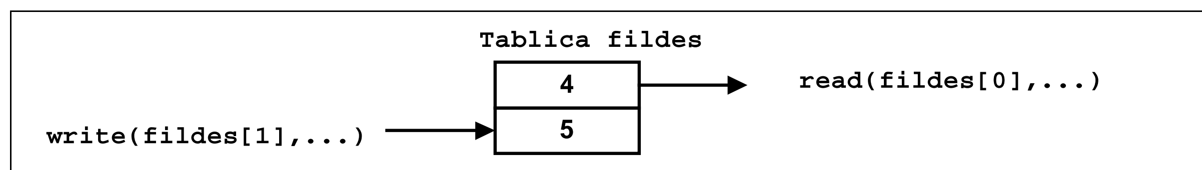
fildes Tablica dwuelementowa na uchwyty plików do odczytu I zapisu

Funkcja tworzy łącze nienazwane i umieszcza w tablicy fildes uchwyty plików:

fildes[0] – uchwyt pliku do odczytu.
fildes[1] – uchwyt pliku do zapisu.

Funkcja zwraca: 0 – sukces, -1 – błąd.

Do pisania i czytania z łącza używa się mechanizmu plików i standardowych funkcji **read** i **write**. Plików nie otwiera się za pomocą funkcji **open**. Stosownych uchwytów dostarcza funkcja **pipe** w tablicy **fildes**.



Rys. 2 Użycie tablicy z uchwytami plików

PDF created with pdfFactory trial version www.pdffactory.com

Własności łącz nienazwanych:

1. Kanał jest jednokierunkowy dla danego procesu i nieużywany w tym procesie plik powinien być zamknięty.
2. Metoda komunikacji może być użyta tylko dla procesów związanych – będących w relacji macierzysty / potomny.
3. Jako że łącze jest buforem typu FIFO utrzymywanym w pamięci operacyjnej ma ono ograniczoną pojemność.
4. Operacje zapisu odczytu do łącza są operacjami atomowymi.

```
main() {
  int fd[2],rd,wr,i;
  char c;
  // Utworzenie łącza ---------------
  if (pipe(fd) < 0) {
      perror("Łącze ");
      exit(10);
  }
  // Utworzenie procesu potomnego ---
  if (fork() > 0) { // Proces macierzysty - czyta z łącza
     close(fd[1]);
     do {
       rd = read(fd[0], &c, 1);
       printf("Odczyt-> %c \n",c);
     } while(rd > 0);
     close(fd[0]);
  } else {  // Proces potomny - pisze do łącza -----------
       close(fd[0]);
       for(i=0;i<10;i++) {
         c= '0' + i;
         printf("Zapis-> %c \n",c);
         write(fd[1], &c,1);
         sleep(1);
       }
       close(fd[1]);
  }
}
```

Przykład 4-2 Przykład komunikacji poprzez łącze nienazwane

PDF created with pdfFactory trial version www.pdffactory.com

Blokowanie odczytu i zapisu przy operowaniu na łączach nienazwanych

Przy posługiwaniu się mechanizmem łącz pojawiają się wątpliwości.

1. Jak zachowa się funkcja **read** gdy odczytywane łącze jest puste ?
2. Jak zachowa się funkcja **write** gdy zapisywane łącze jest pełne ?

O zachowaniu się procesów flaga O_NONBLOCK związana z plikami specjalnymi tworzonymi przez funkcję pipe.

Pliki te domyślnie mają wyzerowaną flagę O_NONBLOCK.
Flagę tę można kontrolować przy pomocy funkcji fcntl.
Wielkość bufora łącza można testować wykonując funkcję:

```
fpathconf(fd,_PC_PIPE_BUF)
```

	Flaga O_NONBLOCK wyzerowana	Flaga O_NONBLOCK ustawiona
Odczyt	Funkcja read blokuje proces bieżący gdy łącze puste	Funkcja read zwraca –1 gdy łącze puste
Zapis	Funkcja write blokuje proces bieżący gdy łącze pełne	Funkcja write zwraca –1 gdy łącze pełne

Tab. 4-3 Wpływ flagi O_NONBLOCK na blokowanie się procesów

Flagę O_NONBLOCK testuje się przy pomocy funkcji:

```
fcntl(fd, F_GETFL, O_NONBLOCK)
```

Ustawić ją można za pomocą funkcji:

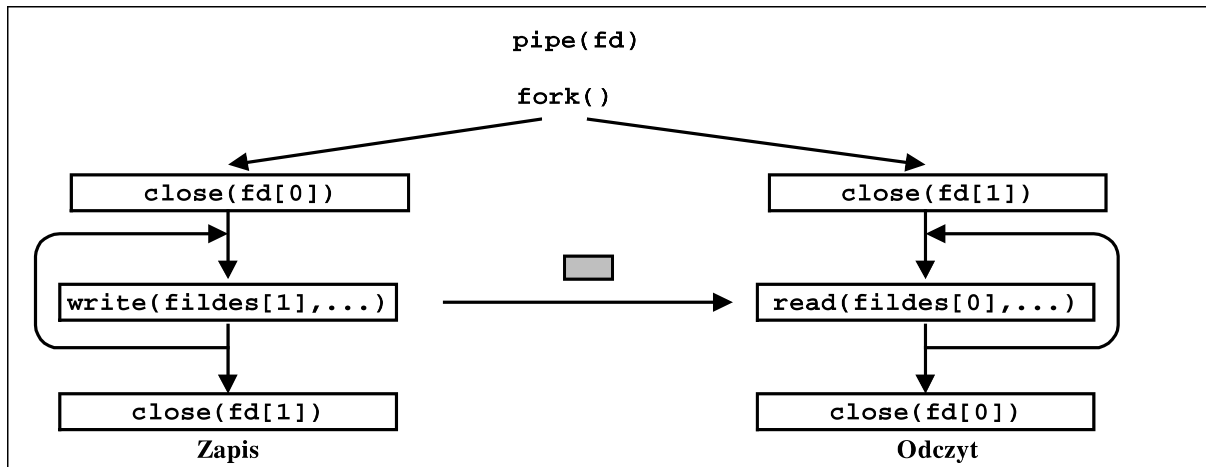
```
fcntl(fd, F_SETFL, fcntl(fd, F_GETFL,O_NONBLOCK)|
O_NONBLOCK)
```

Zamykanie łącz

Co stanie się gdy deskryptor reprezentujący łącze zostanie zamknięty?

1. Zamknięcie deskryptora pliku do zapisu. Gdy istnieją inne procesy które mają otwarte te łącze dla zapisu nie dzieje się nic. Gdy nie gdy nie w łączu danych procesy zablokowane na odczycie zwracają zero.
2. Zamknięcie deskryptora pliku do odczytu. Gdy istnieją inne procesy które mają otwarte te łącze dla odczytu nie dzieje się nic. Gdy nie do wszystkich procesów zablokowanych na zapisie wysłany zostanie sygnał SIGPIPE.

PDF created with pdfFactory trial version www.pdffactory.com



Rys. 3 Wzorzec wykorzystania łącz nienazwanych do komunikacji pomiędzy procesami

PDF created with pdfFactory trial version www.pdffactory.com

4.3 Łącza nazwane – pliki specjalne typu FIFO

Łącza nazwane (ang. named pipes) nazywane też plikami FIFO są narzędziem służącym do komunikacji międzyprocesowej. Posiadają następujące własności:

- tworzone są w pamięci operacyjnej,
- widziane są w przestrzeni nazw plików
- posiadają zwykłe atrybuty pliku w tym prawa dostępu.

Plik FIFO tworzy się przy pomocy funkcji:

```
int mkfifo(char * path, mode_t mode)
```

path Nazwa pliku FIFO (ze ścieżką)
mode Prawa dostępu do pliku .

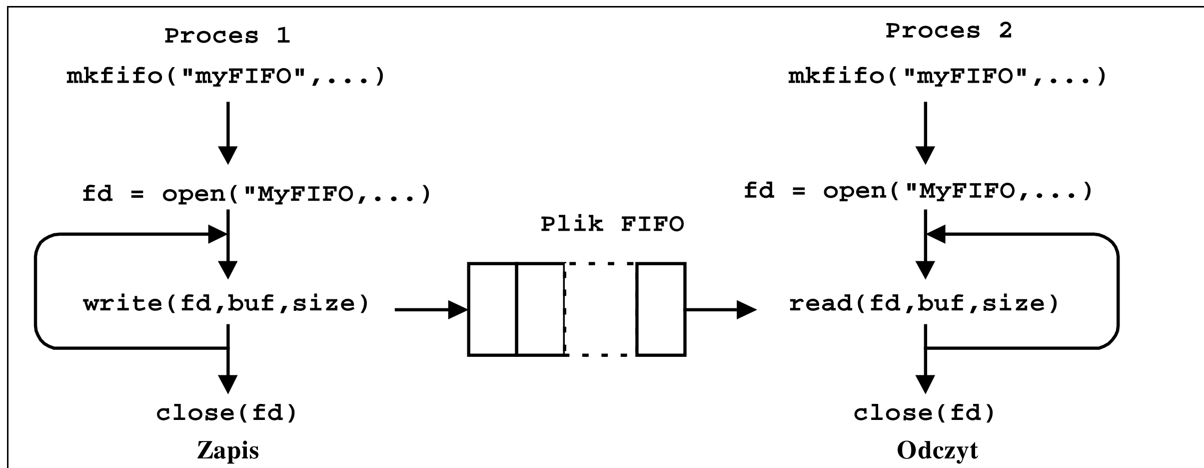
Funkcja zwraca: 0 – sukces, -1 – błąd.

Aby proces mógł użyć pliku FIFO należy:
1. Utworzyć plik FIFO za pomocą funkcji **mkfifo** o ile wcześniej nie został utworzony.
2. Otworzyć plik FIFO za pomocą funkcji **open**.
3. Pisać lub czytać do / z pliku używając funkcji **read** lub **write**.
4. Zamknąć plik przy pomocy funkcji **close**.

Własności plików FIFO.

1. Pliki FIFO są plikami specjalnymi tworzonymi w pamięci operacyjnej ale widzianymi w systemie plików komputera. Stąd procesy mające dostęp do tego samego systemu plików mogą się komunikować przez pliki FIFO.
2. Operacje zapisu odczytu do / z pliku FIFO są operacjami atomowymi.
3. Bajty odczytane z pliku FIFO są stamtąd usuwane.
4. Zachowanie się procesu przy próbie odczytu z pustego pliku FIFO lub zapisu do pełnego zależą od flagi O_NONBLOCK.
5. Informacje w pliku FIFO są pozbawione struktury.
6. Plik FIFO i jego zawartość ginie przy wyłączeniu komputera.

PDF created with pdfFactory trial version www.pdffactory.com



Rys. 4 Wzorzec wykorzystania plików FIFO

```
main() {
  int fdes,res;
  static char c;
  // Utworzenie pliku FIFO ----
  mkfifo("MyPip",S_IRUSR | S_IWUSR);
  fdes = open("MyPip",O_RDONLY);
  if(fdes < 0) { printf("Open error %d \n",errno); exit(1); }
  do {
       res = read(fdes, &c, 1); // Odczyt z pliku FIFO
       printf("Odczytano: %c \n", c);
     } while(res > 0);
     close(fdes);
}
```

Przykład 4-3 Przykład procesu odczytującego znaki z pliku FIFO.

PDF created with pdfFactory trial version www.pdffactory.com

4.4 Sprawdzanie gotowości deskryptorów – funkcja select

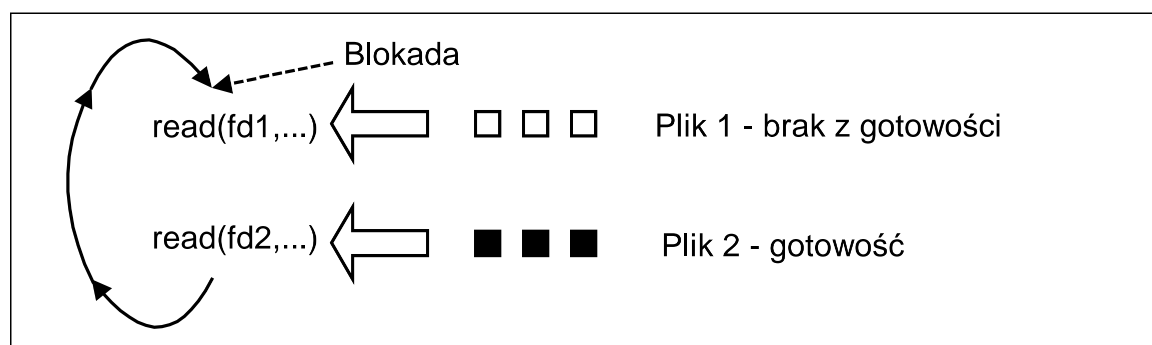
Programy często korzystają z „deskryptorowych” funkcji wejścia-wyjścia. Źródła dające się opisać jako deskryptory plików są nastepujące:

1. Łącza nienazwane (*pipes*)
2. Łącza nazwane (pliki FIFO)
3. Gniazdka (*sockets*)
4. Znakowe rządzenia we/wy - klawiatura, złącza transmisji szeregowej
5. Blokowe urządzenia we/wy – pliki

W sytuacji gdy:

- Proces otrzymuje dane z wielu źródeł
- Odczyt z tych źródeł jest blokujący

Występują trudności z odbiorem danych. Gdy oczekujemy na wejście z jednego procesu (wywołanie blokujące) nie odbieramy tego co jest na innych wejściach i odwrotnie.



Rys. 5 Blokujący odczyt z jednego pliku (fd1) nie pozwala na odczyt danych z drugiego pliku (fd2).

Rozwiązania problemu odczytu (zapisu) z wielu źródeł dostarcza funkcja select.

Funkcja select

Funkcja `select` powoduje zablokowanie procesu bieżącego do czasu wystąpienia gotowości lub błędu na którymś z deskryptorów. Zwraca wtedy numer tego deskryptora. Odblokowuje się również wtedy gdy upłynie zadany okres oczekiwania (ang. *timeout*).

```
#include <sys/time.h>
```

PDF created with pdfFactory trial version www.pdffactory.com

```
int select(int nfds, fd_set *readfds, fd_set
           *writefds, fd_set *errorfds,
           struct timeval * timeout)
```

nfds	Liczba deskryptorów plików (maksymalne FD_SETSIZE)
readfds	Maska deskryptorów plików do odczytu (gotowość odczytu)
writefds	Maska deskryptorów plików do zapisu (gotowość zapisu)
errorfds	Maska deskryptorów plików dotyczących błędów
timeout	Maksymalny okres zablokowania

Po wykonaniu funkcja ponownie ustawia maski bitowe **readfds**, **writefds, errorfds** zgodnie z wynikiem operacji czyli jeden bit ustawia inne zeruje.

Funkcja zwraca:
> 0 – numer deskryptora na którym wystapiła gotowość
 0 – gdy zakończenie na przeterminowaniu
-1 – gdy błąd

Funkcje operujące na maskach bitowych

fd_set - typ zdefiniowany w <sys/time.h>. Bit i ustawiony na 1 o obecności deskryptora i w zbiorze.

Przykład
Deskryptory o numerach 1,2,3 są w zbiorze

7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0

- Zerowanie zbioru fdset

```
void  FD_ZERO( fd_set  *fdset)
```

- Włączenie deskryptora fd do zbioru fdset

```
void  FD_SET( int fd, fd_set  *fdset)
```

- Testowanie czy fd włączony do zbioru fdset

```
int  FD_ISSET( int fd, fd_set  *fdset)
```

- Wyłączenie fd ze zbioru fdset

```
void  FD_CLR( int fd, fd_set  *fdset)
```

PDF created with pdfFactory trial version www.pdffactory.com

```
int fd, fd1, fd2
fd_set  we, we1;
// Otwarcie portów -----------
fd1 = open("/dev/ser1",O_RDWR);
fd2 = open("/dev/ser2",O_RDWR);
// Ustawienie masek bitowych --
FD_ZERO(&we);
FD_SET(fd1,&we);
FD_SET(fd2,&we);

do {
     we1 = we
     fd = select(5,&we1,NULL,NULL,NULL);
     if(FD_ISSET(fd1,&we1) {   read(fd1,...);        }
if(FD_ISSET(fd2,&we1) {   read(fd2,..   .);       }
} while (1);
```

Przykład 4-4 Odczyt znaków z dwóch portów szeregowych

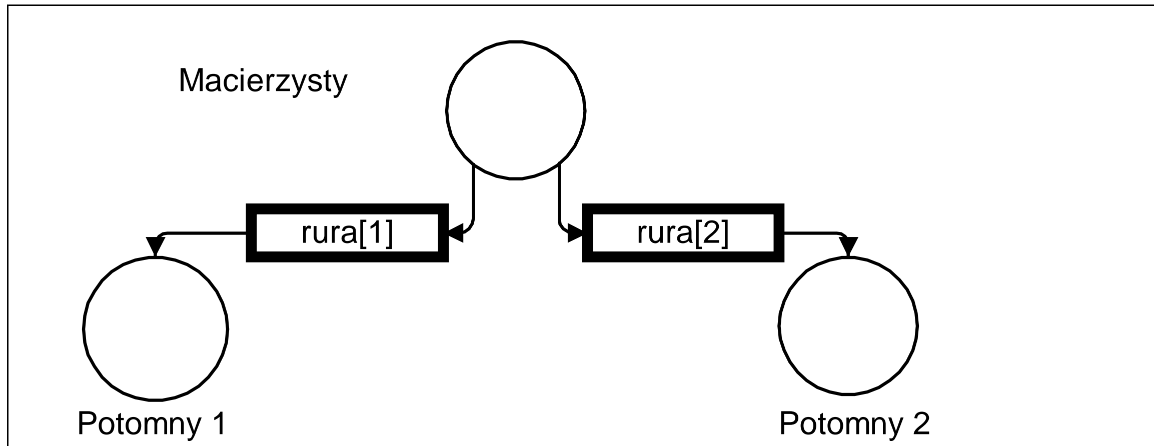
Specyfikacja czasu
Do określenia czasu oczekiwania stosuje się zmienną typu timeval (struktura) zdefiniowaną w pliku **<sys/time.h>.**

```
#include <sys/time.h>
struct timeval {
     long tv_sec;   // sekundy
     long tv_usec;  // mikrosekundy
}

struct timeval tim;
tim.tv_sec = 10;
tim.tv_usec = 0;
```

Przykład 4-5 Ustawienie parametrów przeterminowania

PDF created with pdfFactory trial version www.pdffactory.com

Przykład
Serwer odczytuje zlecenia od 2 procesów klientów przez łącza nienazwane. Nie wiadomo na którym z łącz pojawi się zlecenie.



Rys. 6 Serwer otrzymuje komunikaty z dwóch łącz nienazwanych

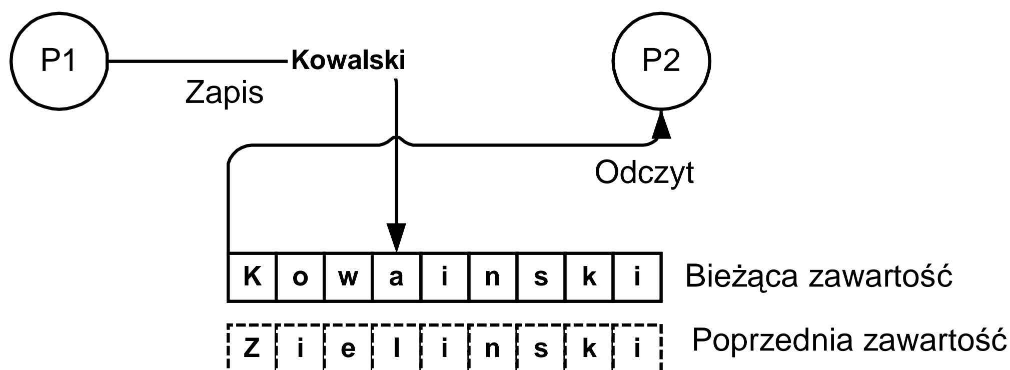
PDF created with pdfFactory trial version www.pdffactory.com

```
#include <sys/select.h>
#define SIZE 9
char msg[2][SIZE] = {"Proces 1","Proces 2"};

void main(void) {
 int rura[2][2];
 int i,pid,numer,bajtow, j = 0;;
 fd_set set;
 char buf[SIZE];

 FD_ZERO(&set);
 printf("set: %x\n",set);
 for(i=0;i<2;i++) {
    pipe(rura[i]);
    FD_SET(rura[i][0],&set);
 }

 for(i=0;i<2;i++) { // Uruchamianie procesów potomnych
   if((pid = fork(0)) == 0) {
    potom(i,rura[i]);
    exit(0);
   }
 }

 // Macierzysty -------------
 do {
  numer = select(FD_SETSIZE,&set,NULL,NULL,NULL);
  for(i=0;i<2;i++) {
    if(FD_ISSET(rura[i][0],&set)) {
       bajtow = read(rura[i][0],buf,SIZE);
       printf("Z: %d otrzymano: %s\n",i,buf);
    }
  }
  j++;
 } while(j<5);
}

int potom(int nr,int rura[2]) {
  int i;
  for(i=0;i<5;i++) {
     printf("Potomny: %d pisze: %s do: %d\n",nr, msg[nr],
             rura[1]);
     write(rura[1],msg[nr],SIZE);
     sleep(1);
  }
  return(1);
}
```

PDF created with pdfFactory trial version www.pdffactory.com

4.5 Komunikacja przez wspólne pliki

Współdzielone pliki mogą być wykorzystane do komunikacji międzyprocesowej. Należy zabezpieczyć się przed jednoczesnym dostępem do pliku przez kilka procesów.



Rys. 7 Współbieżny dostęp do dzielonego pliku powoduje błędny odczyt (zjawisko wyścigów)

Do blokowania dostępu do pliku służy funkcja **`lockf`**:

`int lockf (int` ***`fd`*****`, int`** ***`function`*****`, long`** ***`size`*****`);`**

Gdzie:

fd	deskryptor pliku,
function	F_LOCK - zajmij obszar z blokowaniem, F_ULOCK - zwolnij obszar , F_TLOCK – sprawdź i zajmij obszar gdy jest wolny, gdy zajęty funkcja zwróci błąd. F_TEST - sprawdź czy obszar jest zajęty.
size	rozmiar obszaru do blokady, jeśli 0 to blokowany rekord będzie miał rozmiar od pozycji bieżącej do końca pliku

Funkcja zwraca:
0 gdy zakończy się sukcesem,
-1 gdy błąd.

Proces próbuje zająć określony obszar pliku poczynając od pozycji bieżącej. Gdy plik jest już zajęty proces jest blokowany do jego uwolnienia.

PDF created with pdfFactory trial version www.pdffactory.com

```
#include <fcntl.h>
#include <unistd.h>

int fd;
int status,res;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = lockf(fd, F_TLOCK,0);
res    = write(fd,buf,sizeof(buf));
status = lockf(fd, F_ULOCK,0);
...
```

Przykład 4-6 Blokowanie pliku

Inne funkcje blokowania dostępu do pliku:

- **`flock`** – działa na deskryptorach FILE *
- **`fcntl`** – działa na uchwytach int fd

Gdy operujemy na deskryptorach FILE * można przejść na uchwyty za pomocą funkcji:

```
int fileno(FILE *file)
```

PDF created with pdfFactory trial version www.pdffactory.com