9. Komunikacja przez pamięć dzieloną

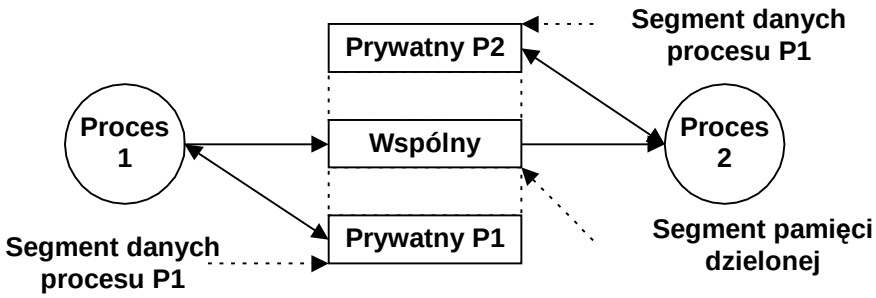
Metoda komunikacji przez wspólną pamięć może być użyta gdy procesy wykonywane są na maszynie jednoprocesorowej lub wieloprocesorowej ze wspólną pamięcią.

Procesy

Procesy mają rozdzielone segmenty danych - modyfikacje wykonane na danych w jednym procesie w żaden sposób nie przenoszą się do procesu drugiego.

Aby procesy mogły mieć wspólny dostęp do tych samych danych należy:

1. Utworzyć oddzielny segment pamięci.
2. Udostępnić dostęp do segmentu zainteresowanym procesom.



Rys. 1 Procesy P1 i P2 komunikuję się poprzez wspólny obszar pamięci

Wątki

Wątki z natury dzielą obszar danych. Zmienne zadeklarowane jako zmienne globalne będą dostępne dla wątków.

Komunikacja poprzez pamięć dzieloną

Gdy procesy komunikują się przez wspólną pamięć, należy zadbać o zachowania spójności danych zawartych w dzielonym obszarze pamięci.

9.1 Funkcje operujące na wspólnej pamięci – standard Posix

Standard Posix 1003.4 - funkcje pozwalające na tworzenie i udostępnianie segmentów pamięci:

Działanie	Funkcja
Utworzenie wspólnego segmentu pamięci	**shm_open()**
Ustalenie rozmiaru segmentu	**ftruncate()**
Ustalenie odwzorowanie segmentu	**map()**
Cofnięcie odwzorowania segmentu	**munmap()**
Zmiana trybu dostępu	**mprotect()**
Skasowanie segmentu pamięci	**shm_unlink()**

Tabela 9-1 Funkcje POSIX operujące na pamięci wspólnej

Tworzenie segmentu pamięci

Tworzenie segmentu pamięci podobne jest do tworzenia pliku – segment jest plikiem specjalnym.

```
int shm_open(char *name, int oflag, mode_t mode )
```

name Nazwa segmentu pamięci
oflag Flaga specyfikująca tryb utworzenia (jak dla plików), np. O_RDONLY, O_RDWR, O_CREAT
mode Specyfikacja trybu dostępu (jak dla plików).

Gdy funkcja zwraca liczbę nieujemną jest to uchwyt identyfikujący segment w procesie. Segment widziany jest jako plik specjalny w katalogu /dev/shmem.

Ustalanie rozmiaru segmentu pamięci

```
off_t ltrunc(int fdes, off_t offset, int whence)
```

fdes Uchwyt segmentu zwracany przez poprzednią funkcję shm_open.
offset Wielkość segmentu w bajtach.
whence W tym przypadku stała SEEK_SET

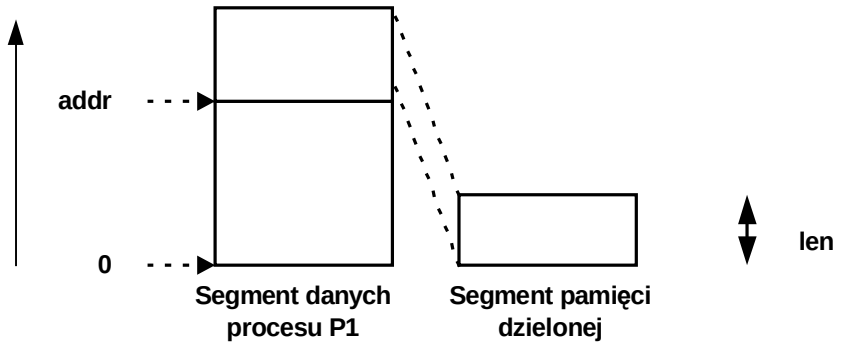
Funkcja zwraca wielkość segmentu lub –1 gdy błąd.

Odwzorowanie segmentu pamięci wspólnej w obszar procesu,

```
void *mmap(void * addr, size_t len, int prot, int flags,
int fdes, off_t off)
```

addr Zmienna wskaźnikowa w procesie której wartość będzie przez funkcję zainicjowana. Może być 0.
len Wielkość odwzorowywanego obszaru.
prot Specyfikacja dostępu do obszaru opisana w <sys/mman.h>. Może być PROT_READ|PROT_WRITE
flags Specyfikacja użycia segmentu, np. MAP_SHARED.
fdes Uchwyt segmentu wspólnej pamięci.
off Początek obszaru we wspólnej pamięci (musi to być wielokrotność strony 4K)

Funkcja zwraca adres odwzorowanego obszaru lub –1 gdy błąd.



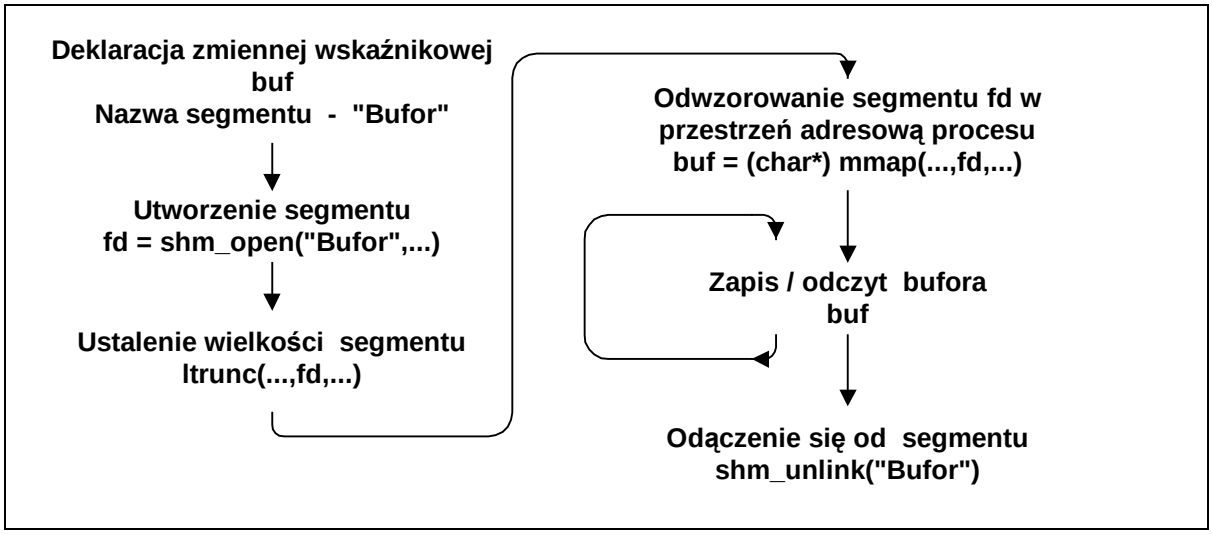
Odłączenie się od segmentu pamięci

```
shm_unlink(char *name)
```

name Nazwa segmentu pamięci.

Każde wywołanie tej funkcji zmniejsza licznik udostępnień segmentu. Gdy osiągnie on wartość 0 czyli segment nie jest używany już przez żaden proces, segment jest kasowany.

Schemat utworzenia i udostępnienia segmentu podano na poniższym rysunku.



Schemat użycia segmentu pamięci dzielonej

Sposób wykorzystania wspólnej pamięci do komunikacji pomiędzy procesami.

Proces macierzysty P1:

1. Deklaruje zmienną wskaźnikową buf.
2. Tworzy segment pamięci o nazwie „Bufor” - funkcja shm_open.
3. Ustala jego wielkość na B_SIZE - funkcją ltrunc.
4. Udostępnia segment w przestrzeni adresowej inicjując zmienną buf – funkcja mmap.
5. Tworzy proces potomny P2 – funkcja fork.
6. Czyta znaki z bufora buf.
7. Odłącza się od segmentu – funkcja shm_unlink.

Proces potomny P2:

1. Korzysta z utworzonego, udostępnionego i odwzorowanego jako buf segmentu pamięci.
2. Pisze znaki do bufora buf.

```
#include <sys/mman.h>
#define B_SIZE 60   // Rozmiar bufora
#define STEPS  6    // Liczba krokow
main(int argc,char *argv[]) {
 int i,stat;
 char  *buf;
 char  *tbuf;
 char name[16];
 char c;
 int  fd; // Deskryptor segmentu

 strcpy(name,"Bufor");
 // Allokacja pamieci dzielonej na bufor ------------
 // Utworzenie segmentu pamieci  --------------------
 if((fd=shm_open(name,O_RDWR|O_CREAT,0664))==-1)exit(-1)

 // Okreslenie rozmiaru obszaru pamieci --------------
 if(ltrunc(fd, B_SIZE, SEEK_SET) != B_SIZE) exit(-1);

 // Odwzorowanie segmentu fd w obszar pamieci procesow
 buf = (char *)mmap(0,B_SIZE, PROT_READ|PROT_WRITE,
        MAP_SHARED, fd, 0);
 if(buf == NULL) exit(-1);
 // Proces potomny P2 - pisze do pamieci wspolnej -----
 if(fork() == 0) {    for(i=0;i<STEPS;i++) {
     tbuf = buf;
     for(k=0;k<B_SIZE;k++) { // Zapis do bufora
        *(tbuf++) = '0'+ (i%10);
     }
   }
   exit(0);
 } else {
 // Proces macierzysty P1 – czyta z pamięci wspólnej -
    for(i=0;i<STEPS;i++) {
     tbuf = buf;
     for(k=0;k<B_SIZE;k++) {
       c = *(tbuf++);
       putch(c);
     }
   }
 }
 pid = wait(&stat);
 // Zwolnienie nazwy -----
 shm_unlink(name);
 return 0;
}
```

Przykład 1 Procesy P1 i P2 komunikują się przez wspólny obszar pamięci

9.2 Funkcje operujące na wspólnej pamięci – IPC UNIX System V

Klucze mechanizmów IPC

Mechanizmy komunikacji międzyprocesowej (kolejki komunikatów, semafory, segmenty pamięci dzielonej) wymagają identyfikacji tych obiektów w obrębie pojedynczego komputera. Identyfikacja ta odbywa się za pomocą kluczy (*ang. key*).

Do otrzymywania unikalnych kluczy generowanych na podstawie ścieżki do pliku służy funkcja ftok.

key_t ftok(char *path, int id);

Gdzie:

path	Scieżka do pliku
Id	Numer identyfikacyjny

Funkcja zwraca klucz zależny od parametrów path, id.

Tworzenie segmentu wspólnej pamięci

Jeżeli mamy korzystać z segmentu pamięci wspólnej należy go najpierw utworzyć. Segment tworzy się przy pomocy funkcji shmget.

```
int shmget(key_t key, size_t size, int flags)
```

Key	Klucz identyfikujący semafor
Size	Minimalny rozmiar obszaru wspólnego
Flags	Flagi specyfikujące tryb tworzenia i prawa dostępu (IP_CREAT, IPC_EXCL, SHM_RND, SHM_RDONLY)

Funkcja zwraca:
> 0 identyfikator segmentu wspólnego
-1 błąd

Dołączenie segmentu pamięci

Po utworzeniu segment pamięci wspólnej musi być udostępniony w przestrzeni adresowej procesu który ma go używać.
Udostępnienie to następuje przez wykonanie funkcji shmat.

```
void *shmat(int id, void addr , int flags)
```

Gdzie:

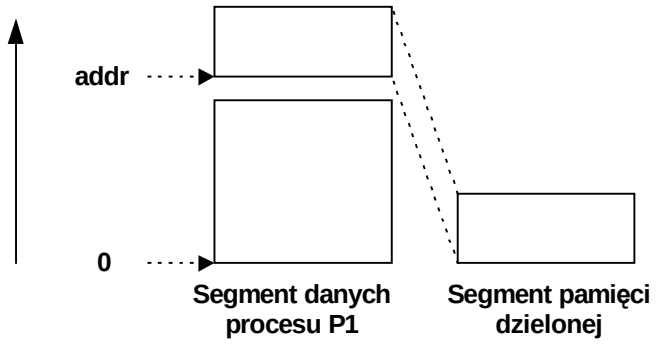
Id	identyfikator segmentu zwracany przez funkcję shmget
addr	Zmienna specyfikująca adres obszaru wspólnego
flags	Flagi specyfikujące tryb dostępu i przydzielania adresu

Funkcja zwraca:
> 0 - adres przydzielonego segmentu wspólnego
-1 - błąd

Gdy wartość zmiennej addr jest równa NULL system sam nada wartość adresowi po którym widziany jest dołączany segment pamięci. Gdy nie jest równy zeru działanie funkcji zależy od ustawienia flag.

Gdy flaga SHM_RND jest ustawiona system przyjmie adres pod którym ma być widziany segment wspólny jako addr ale zaokrągli go do najbliższej strony.

Gdy flaga SHM_RND jest wyzerowana system ustali adres pod którym ma być widziany segment wspólny jako dokładna wartość addr.
Inną flagą mającą tu zastosowanie jest flaga SHM_RDONLY. Gdy jest ona ustawiona segment wspólny nadaje się tylko do odczytu.



Rys. 2 Odwzorowanie segmentu pamięci dzielonej w przestrzeni adresowej procesu

Dołączony za pomocą funkcji shmat segment pamięci wspólnej można odłączyć używając funkcji shmdt.

```
int shmdt(void *addr)
```

Parametr addr jest wartością zwróconą przez funkcję shmat. Funkcja shmdt zwraca 0 gdy wykona się poprawnie a –1 w gdy wystąpi błąd.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/shm.h>

#define SIZE 8

main(int argc, char *argv[]) {
    int d, i;
    int *x;
    d = shmget(ftok(argv[0], 1),
                sizeof(int) * SIZE, IPC_CREAT | 0600);
    x = shmat(d, NULL, 0);

    for (;;) {
        for (i = 0; i < SIZE; i++)
            printf("%3d ", x[i]);
        printf("\n");
        for(i=0; i < SIZE;i++) x[i]++;
        sleep(1);
    }
}
```

```
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2
```

Przykład 2 Wykonanie pojedynczej kopii programu shmem1

```
0 0 0 0 0 0 0 0
2 2 2 2 2 2 2 2
4 4 4 4 4 4 4 4
```

Przykład 3 Wykonanie dwóch kopii programu shmem1

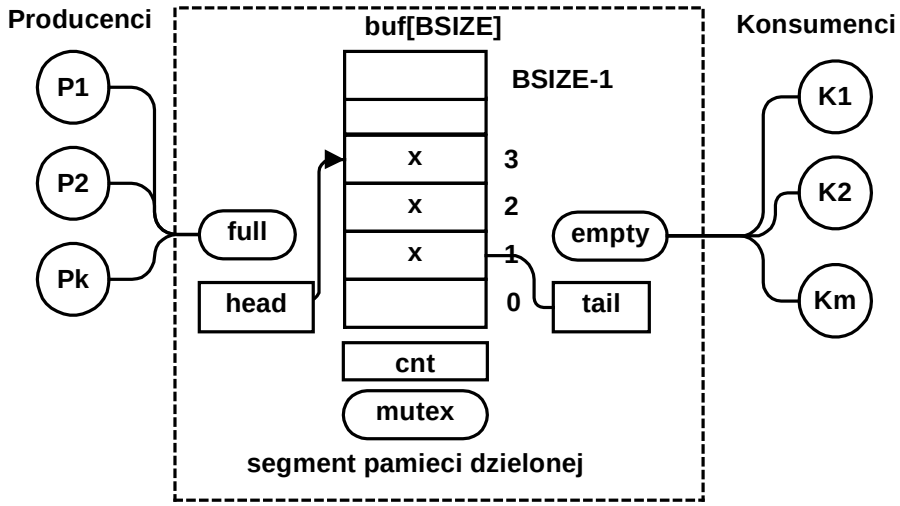
```
// Producent – konsument (rozwiązanie bledne)
#include <stdio.h>
#include <unistd.h>
#include <sys/shm.h>

#define TSIZE 64
typedef struct {
        int licznik;
        char tekst[TSIZE];
       } buft;

main(int argc, char *argv[]) {
    int d, i;
    buft *buf;
    d = shmget(ftok(argv[0], 1),
                sizeof(buft), IPC_CREAT | 0600);
    buf = shmat(d, NULL, 0);
    if(argc < 2) {
       printf("Uzycie shmpk [P|K]\n");
       exit(0);
    }
    if(argv[1][0] == 'P') { // Producent
       printf("Producent\n");
       for (;;) {
        while(buf->licznik > 0); // wait
            sprintf(buf->tekst,"Producent krok %d ",i);
            buf->licznik++;
            i++;
            sleep(1);
       }
     }
     if(argv[1][0] == 'K') { // Konsument
       printf("Konsument\n");
       for (;;) {
        while(buf->licznik == 0); // wait
            printf("%s\n",buf->tekst);
            buf->licznik--;
            sleep(1);
       }
     }
     memset(buf,0,sizeof(buft));
     printf("Uzycie shmpk [P|K]\n");
}
```

Przykład 4 Problem producenta / konsumenta (rozw. nieprawidłowe)

9.3 Rozwiązanie problemu producenta i konsumenta – semafory nienazwane




```
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#define BSIZE      4   // Rozmiar bufora
#define LSIZE    80   //  Dlugosc linii
typedef struct {
    char buf[BSIZE][LSIZE];
    int head;
    int tail;
    int cnt;
    sem_t mutex;
    sem_t empty;
    sem_t full;
} bufor_t;

main(int argc,char *argv[]) {
 int i,stat,k, pid, size, fd,res;
 bufor_t  *wbuf ;
 char c;
 // Utworzenie segmentu --------------------------
 shm_unlink("bufor");
 if((fd=shm_open("bufor", O_RDWR|O_CREAT , 0774)) == -1){
   perror("open"); exit(-1);
 }
 printf("fd: %d\n",fd);
 size = ftruncate(fd, BSIZE);
 if(size < 0) {perror("trunc"); exit(-1); }
```

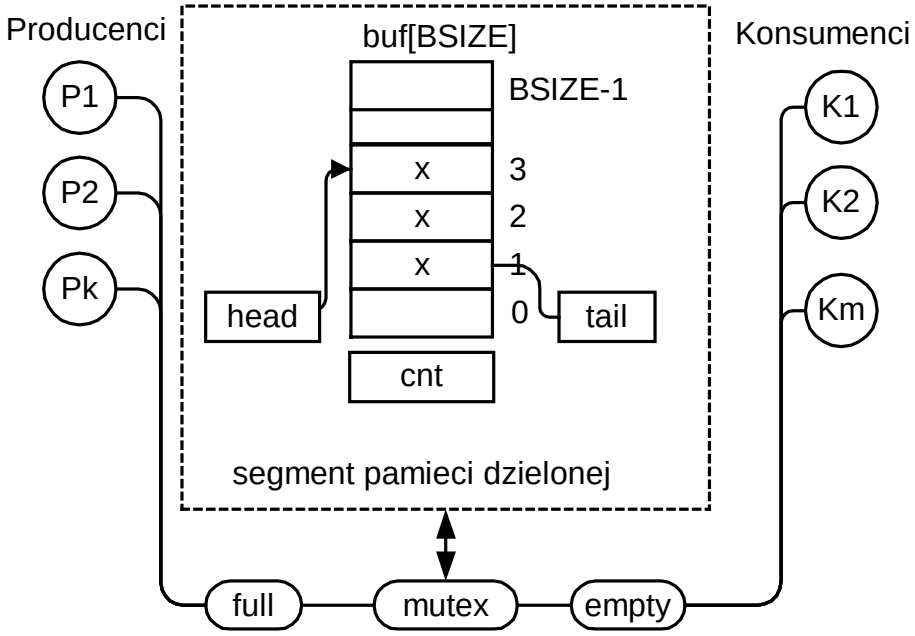
```
// Odwzorowanie segmentu fd w obszar pamieci procesow
wbuf = (bufor_t *)mmap(0,BSIZE,PROT_READ|PROT_WRITE,
        MAP_SHARED, fd, 0);
if(wbuf == NULL) {perror("map");  exit(-1); }

// Inicjacja obszaru -------------------------------
wbuf-> cnt = 0;
wbuf->head = 0;
wbuf->tail = 0;
if(sem_init(&(wbuf->mutex),1,1)){
  perror("mutex");exit(0);
}
if(sem_init(&(wbuf->empty),1,BSIZE)) {
  perror("empty"); exit(0);
}
if(sem_init(&(wbuf->full),1,0)) {
  perror("full"); exit(0);
}
// Tworzenie procesow -------------
if(fork() == 0) { // Producent
   for(i=0;i<10;i++) {
      // printf("Producent: %i\n",i);
      printf("Producent - cnt:%d head: %d tail: %d\n",
              wbuf-> cnt,wbuf->head,wbuf->tail);
      sem_wait(&(wbuf->empty));
      sem_wait(&(wbuf->mutex));
      sprintf(wbuf->buf[wbuf->head],"Komunikat %d",i);
      wbuf-> cnt ++;
      wbuf->head = (wbuf->head +1) % BSIZE;
      sem_post(&(wbuf->mutex));
      sem_post(&(wbuf->full));
      sleep(1);
    }
    shm_unlink("bufor");
    exit(i);
}
 // Konsument ------------------
 for(i=0;i<10;i++) {
    printf("Konsument - cnt: %d odebrano %s\n",wbuf->cnt
            ,wbuf->buf[wbuf->tail]);
    sem_wait(&(wbuf->full));
    sem_wait(&(wbuf->mutex));
    wbuf-> cnt --;
    wbuf->tail = (wbuf->tail +1) % BSIZE;
    sem_post(&(wbuf->mutex));
    sem_post(&(wbuf->empty));
    sleep(1);
}
```

```
 pid = wait(&stat);
 shm_unlink("bufor");
 sem_close(&(wbuf->mutex));
 sem_close(&(wbuf->empty));
 sem_close(&(wbuf->full));
 return 0;
}
```

Przykład 9-5 Rozwiązanie problemu producenta i konsumenta za pomocą semaforów nienazwanych

9.4 Rozwiązanie problemu producenta i konsumenta – semafory nazwane




```
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#define BSIZE     4  // Rozmiar bufora
#define LSIZE    80  // Dlugosc linii

typedef struct { // Obszar wspólny
    char buf[BSIZE][LSIZE];
    int head;
    int tail;
    int cnt;
} bufor_t;

main(int argc,char *argv[]) {
 int i,stat,k, pid, size, fd,res;
 bufor_t  *wbuf ;
 char c;
 sem_t *mutex;
 sem_t *empty;
 sem_t *full;

 // Utworzenie segmentu --------------------------
 shm_unlink("bufor");
 if((fd=shm_open("bufor", O_RDWR|O_CREAT , 0774)) == -1){
   perror("open"); exit(-1);
 }
 printf("fd: %d\n",fd);
```

```
size = ftruncate(fd, BSIZE);
if(size < 0) {perror("trunc"); exit(-1); }
// Odwzorowanie segmentu fd w obszar pamieci procesow
wbuf = ( bufor_t *)mmap(0,BSIZE, PROT_READ|PROT_WRITE,
        MAP_SHARED, fd, 0);
if(wbuf == NULL) {perror("map");  exit(-1); }

// Inicjacja obszaru -------------------------------
wbuf-> cnt = 0;
wbuf->head = 0;
wbuf->tail = 0;

// Utworzenie semaforw ------------------------
mutex = sem_open("mutex",O_CREAT,S_IRWXU,1);
empty = sem_open("empty",O_CREAT,S_IRWXU,BSIZE);
full  = sem_open("full",O_CREAT,S_IRWXU,0);

// Utworzenie procesow ------------------------
if(fork() == 0) { // Producent
   for(i=0;i<10;i++) {
     // printf("Producent: %i\n",i);
     sem_wait(empty);
     sem_wait(mutex);
     printf("Producent - cnt:%d head: %d tail: %d\n",
             wbuf-> cnt,wbuf->head,wbuf->tail);
     sprintf(wbuf->buf[wbuf->head],"Komunikat %d",i);
     wbuf-> cnt ++;
     wbuf->head = (wbuf->head +1) % BSIZE;
     sem_post(mutex);
     sem_post(full);
     sleep(1);
   }
   shm_unlink("bufor");
   exit(i);
}
```

```
  // Konsument ------------------
  for(i=0;i<10;i++) {
     sem_wait(full);
     sem_wait(mutex);
     printf("Konsument - cnt: %d odebrano %s\n",
            wbuf->cnt,wbuf->buf[wbuf->tail]);
     wbuf-> cnt --;
     wbuf->tail = (wbuf->tail +1) % BSIZE;
     sem_post(mutex);
     sem_post(empty);
     sleep(1);
 }
 pid = wait(&stat);
 shm_unlink("bufor");
 sem_close(mutex);    sem_close(empty); sem_close(full);
 sem_unlink("mutex"); sem_unlink("empty");
 sem_unlink("full");
 return 0;
}
```

Przykład 9-6 Rozwiązanie problemu producenta i konsumenta za pomocą semaforów nazwanych