

## 9. Komunikacja przez pamięć dzieloną

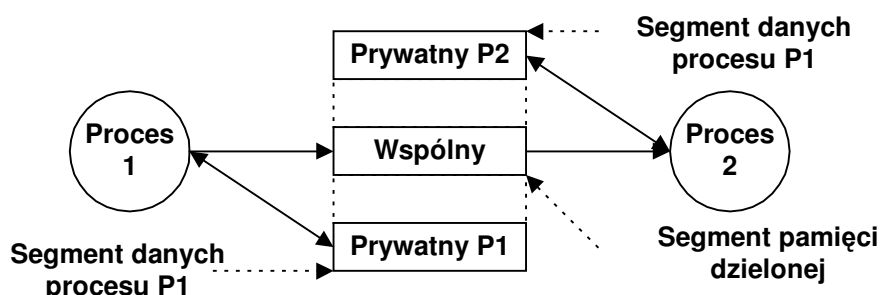
Metoda komunikacji przez wspólną pamięć może być użyta gdy procesy wykonywane są na maszynie jednoprocessorowej lub wieloprocessorowej ze wspólną pamięcią.

### 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. 9-1 Procesy P1 i P2 komunikują 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 zachowanie 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	<b>shm_open()</b>
Ustalenie rozmiaru segmentu	<b>ftruncate()</b>
Ustalenie odwzorowanie segmentu	<b>mmap()</b>
Cofnięcie odwzorowania segmentu	<b>munmap()</b>
Zmiana trybu dostępu	<b>mprotect()</b>
Skasowanie segmentu pamięci	<b>shm_unlink()</b>

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 ftruncate(int fdes, off_t offset)
```

**fdes** Uchwyt segmentu zwracany przez poprzednią funkcję shm\_open.  
**offset** Wielkość segmentu w bajtach.

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.

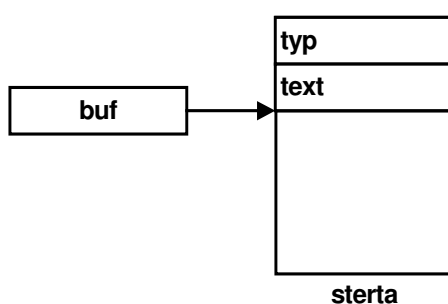
**fdes** Uchwyt segmentu wspólnej pamięci.

**off** Początek obszaru we wspólnej pamięci (musi to być wielokrotność strony 4K)

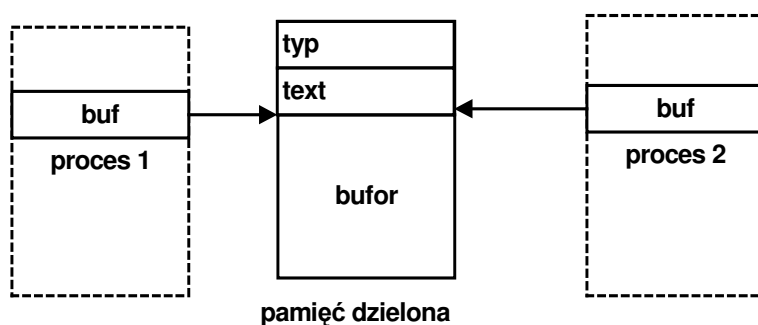
Funkcja zwraca adres odwzorowanego obszaru lub -1 gdy błąd.

```
typedef struct {  
    int typ;  
    char text[SIZE];  
} buf_t;  
...  
buf_t *buf;  
buf = malloc(sizeof(buf_t));  
buf->typ = 1;  
...
```

Przykład 1 Pamięć zawierająca strukturę znajduje się na stercie



Rys. 9-2 Struktura buf na stercie



Rys. 9-3 Struktura buf w pamięci dzielonej

```

typedef struct {
    int typ;
    char text[SIZE];
} buf_t;
...
buf_t *buf;
....
fd=shm_open("bufor",O_RDWR|O_CREAT,0664);
...
buf = (buf_t *)mmap(0,sizeof(buf_t),PROT_READ|PROT_WRITE,
    MAP_SHARED, fd, 0);

buf->typ = 1;
...

```

Przykład 2 Pamięć zawierająca strukturę znajduje się w pamięci dzielonej

### 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.

```

// Kompilacja gcc pam-dziel.c -o pam-dziel -lrt
#include <sys/mman.h>
#include <stdio.h>

```

```
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

#define SIZE 60 // Rozmiar bufora

typedef struct {
    int typ;
    char text[SIZE];
} buf_t;

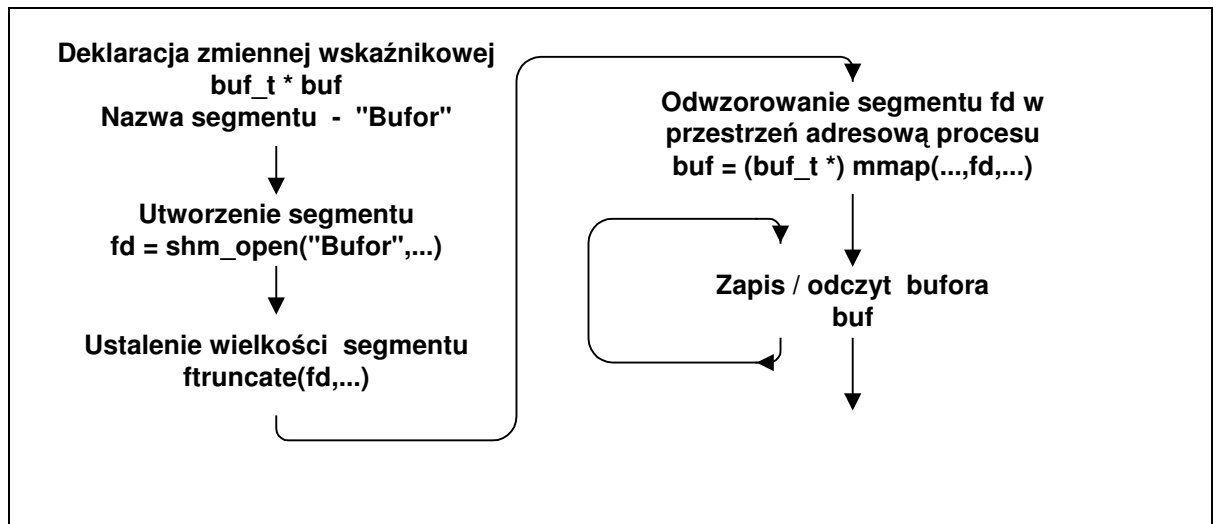
main(int argc, char *argv[]) {
    int i, stat, pid, k, res;
    buf_t *buf;
    char name[16];
    int fd; // Deskryptor segmentu

    strcpy(name, "Bufor");
    shm_unlink(name);
    // Utworzenie segmentu pamieci -----
    if((fd=shm_open(name, O_RDWR|O_CREAT, 0664))==-1) {
        perror("shm_open");
        exit(-1);
    }
    printf("fh = %d\n", fd);
    // Okreslenie rozmiaru obszaru pamieci -----
    res = ftruncate(fd, sizeof(buf_t));
    if(res < 0) { perror("ftrunc"); return 0; }

    // Odwzorowanie segmentu fd w obszar pamieci procesow
    buf = (buf_t *) mmap(0, sizeof(buf_t),
    PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
    if(buf == NULL) { perror("mmap"); exit(-1); }
    printf("start\n");
    // Proces potomny P2 - pisze do pamieci wspolnej -----
    if(fork() == 0) {
        buf-> typ = 1;
        for(k=0; k<10; k++) { // Zapis do bufora
            printf("Zapis - Komunikat %d\n", k);
            sprintf(buf->text, "Komunikat %d", k);
            sleep(1);
        }
        exit(0);
    }
    // Proces macierzysty P1 czyta z pamieci wspolnej -
    for(i=0; i<10; i++) {
        printf("Odczyt %s\n", buf->text);
        sleep(1);
    }
}
```

```
}  
// Czekam na potomny --  
pid = wait(&stat);  
return 0;  
}
```

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



Rys. 9-4 Schemat użycia segmentu pamięci dzielonej

Sposób wykorzystania wspólnej pamięci do komunikacji pomiędzy procesami.

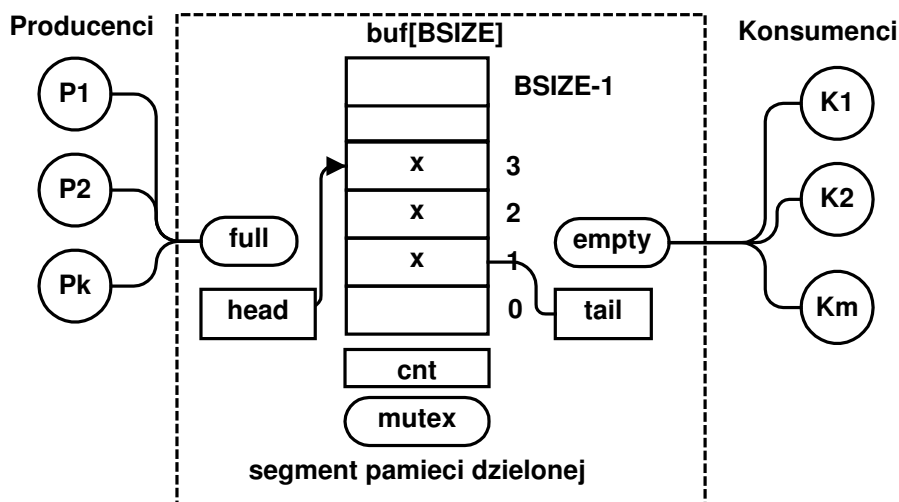
Proces macierzysty P1:

1. Deklaruje zmienną wskaźnikową buf do struktury buf\_t.
2. Tworzy segment pamięci o nazwie „Bufor” - funkcja shm\_open.
3. Ustala jego wielkość na sizeof(buf\_t) - funkcją ftrunc.
4. Udostępnia segment w przestrzeni adresowej inicjując zmienną buf – funkcja mmap.
5. Tworzy proces potomny P2 – funkcja fork.
6. Czyta znaki z bufora buf.

Proces potomny P2:

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

## 9.2 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); }
```



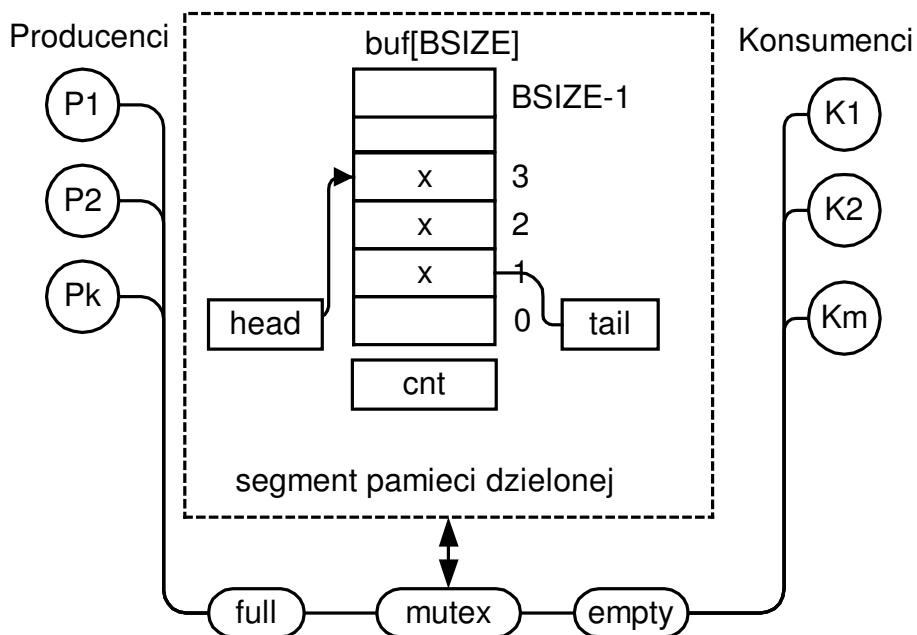
```
// Odzworowanie segmentu fd w obszar pamieci procesow
wbuf = (bufor_t *)mmap(0,BSIZE,PROT_READ|PROT_WRITE,
    MAP_SHARED, fd, 0);
if(wbuf == NULL) {perror("map"); exit(-1); }

// 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 9-2 Rozwiązanie problemu producenta i konsumenta za pomocą semaforów nienazwanych

### 9.3 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 semaforow -----
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 9-3 Rozwiązanie problemu producenta i konsumenta za pomocą semaforów nazwanych