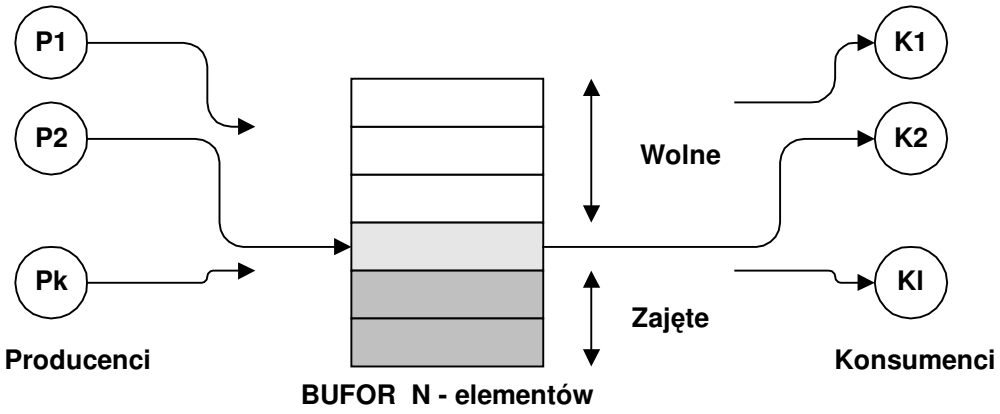
10. Synchronizacja użycia zasobów, Semafory

10.1 Problem producenta i konsumenta

Zagadnienie kontroli użycia jednostek zasobu
W systemie istnieje pula N jednostek zasobu pewnego typu. Procesy mogą pobierać z puli zasoby i je zwracać. Gdy brak jest zasobu a pewien proces będzie próbował go pobrać ulega on zablokowaniu. Proces zostanie odblokowany gdy inny proces zwróci jednostkę zasobu.

1. W systemie istnieją dwie grupy procesów – producenci i konsumenci oraz bufor na elementy.
2. Producenci produkują pewne elementy i umieszczają je w buforze.
3. Konsumenci pobierają elementy z bufora i je konsumują .
4. Producenci i konsumenci przejawiają swą aktywność w nie dających się określić momentach czasu.
5. Bufor ma pojemność na N elementów.



Należy prawidłowo zorganizować pracę systemu.

1. Gdy są wolne miejsca w buforze producent może tam umieścić swój element. Gdy w buforze brak miejsca na elementy producent musi czekać. Gdy wolne miejsca się pojawią producent zostanie odblokowany.
2. Gdy w buforze są jakieś elementy konsument je pobiera. Gdy brak elementów w buforze konsument musi czekać. Gdy jakiś element się pojawi, konsument zostanie odblokowany.

Bufor zorganizowany może być na różnych zasadach.

1. Kolejka FIFO (bufor cykliczny).
2. Kolejka LIFO (stos).

Umieszczanie / pobieranie elementu z bufora jest sekcją krytyczną.

10.2 Semafory

Semafor jest obiektem abstrakcyjnym służącym do kontrolowania dostępu do ograniczonego zasobu. Semafory są szczególnie przydatne w środowisku gdzie wiele procesów lub wątków komunikuje się przez wspólną pamięć.

Definicja semafora.

Semafor S jest obiektem systemu operacyjnego z którym związany jest licznik L zasobu przyjmujący wartości nieujemne. Na semaforze zdefiniowane są *atomowe* operacje **`sem_init`**, **`sem_wait`** i **`sem_post`**.

Podano je w poniższej tabeli.

Operacja	Oznaczenie	Opis
Inicjacja semafora S	sem_init(S,N)	Ustawienie licznika semafora S na początkową wartość N .
Zajmowanie	sem_wait(S)	Gdy licznik L semafora S jest dodatni (L > 0) zmniejsz go o 1 (L = L – 1). Gdy licznik L semafora S jest równy zero (L= 0) zablokuj proces bieżący.
Sygnalizacja	sem_post(S)	Gdy istnieje jakiś proces oczekujący na semaforze S to odblokuj jeden z czekających procesów. Gdy brak procesów oczekujących na semaforze S zwiększ jego licznik L o 1 (L=L+1).

Definicja operacji wykonywanych na semaforze

Uwaga!
1. Semafor nie jest liczbą całkowitą na której można wykonywać operacje arytmetyczne .
2. Operacje na semaforach są operacjami atomowymi.

```
sem_wait(S) {
 if(Licznik L sem. S dodatni){
  L=L-1;
 } else {
  Zawieś proces bieżący
 }
}
```

```
sem_post (S) {
 if(Istnieje proc. czekający na
zasób) {
Odblokuj jeden z tych procesów
} else {
    L=L+1;
  }
}
```

Niezmiennik semafora

Aktualna wartość licznika L semafora S spełnia następujące warunki:

1. Jest nieujemna czyli: L >= 0
2. Jego wartość wynosi: L= N - Liczba_operacji(sem_wait) + Liczba_operacji(sem_post). (N jest wartością początkową licznika).

Semafor binarny

W semaforze binarnym wartość licznika przyjmuje tylko dwie wartości: 0 i 1.

Rodzaje semaforów

Wyróżniamy następujące rodzaje semaforów:

1. Semafor ze zbiorem procesów oczekujących (*ang. Blocked- set Semaphore*) – Nie jest określone który z oczekujących procesów ma być wznowiony.

2. Semafor z kolejką procesów oczekujących (*ang. Blocked-queue Semaphore*) – Procesy oczekujące na semaforze umieszczone są w kolejce FIFO.

Uwaga!
Pierwszy typ semafora nie zapewnia spełnienia warunku zagłodzenia.

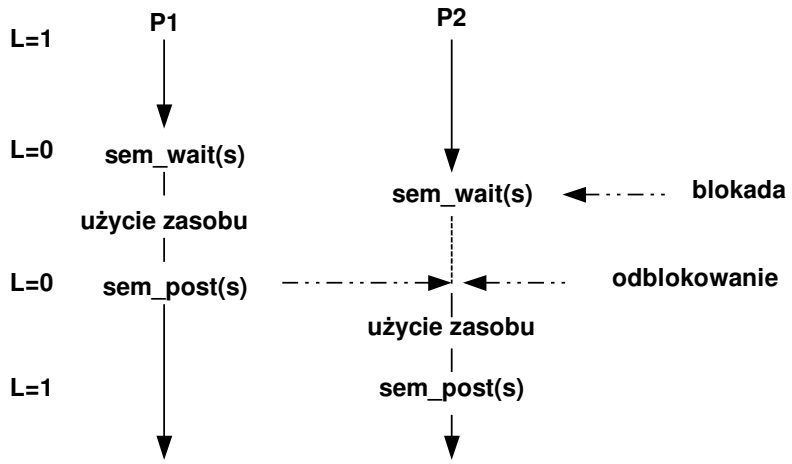
10.3 Zastosowania semaforów

Implementacja wzajemnego wykluczania

```
sem_t S;                          // Deklaracja semafora
sem_init(S,1);                    // Inicjacja semafora

Proces1 {                         Proces2 {
 do {                              do {
   Sekcja_lokalna;                   Sekcja_lokalna;
   sem_wait(S);                      sem_wait(S);
   Sekcja_krytyczna;                 Sekcja_krytyczna;
   sem_post(S);                      sem_post(S);
 } while(1);                       } while(1);
}                                 }
```

Program 10-1 Przykład zastosowania semafora do ochrony sekcji krytycznej



Rys. 10-1 Przebieg operacji blokowania

Rozwiązanie problemu producenta – konsumenta

```
#define BufSize 8                    // Bufor ma 8 elementów
RecType Buffer[BufSize];             // Bufor na elementy
sem_t   Mutex;                       // Ochrona bufora
sem_t   Puste;                       // Wolne bufory
sem_t   Pelne;                       // Zajete bufory
int     count;                       // Wskaźnik bufora

// Kod wątku producenta ---------- // Kod wątku konsumenta -------
producent(void) {                    konsument(void) {
 RecType x;                           RecType x;
 do {                                 do {
   ...                                  ...
  produkcja rekordu x;                  ...
  // Czekaj na wolny bufor             // Czekaj na element
  sem_wait(Puste);                     sem_wait(Pelne);
  sem_wait(Mutex);                     sem_wait(Mutex);
    // Umieść element x w buforze      // Pobierz element x z bufora
    Buffer[count] = x;                 count--;
    count++;                           x = Buffer[count];
  sem_post(Mutex);                     sem_post(Mutex);
  // Pojawił się nowy element          // Zwolnij miejsce w buforze
  sem_post(Pelne);                     sem_post(Puste);
                                       konsumpcja rekordu x;
 } while(1);                             ...
}                                     } while(1);
                                     }
main(void) {
 count = 0;
 sem_init(Puste,BufSize);            // Inicjacja semafora Puste
 sem_init(Pelne,0);                  // Inicjacja semafora Pelne
 sem_init(Mutex,1);                  // Inicjacja semafora Mutex
 pthread_create(…,producent,..);     // Start K wątków producenta
 ..
 pthread_create(…,konsument,..);     // Start L wątków konsumenta
  ..
}
```

Program 10-2 Rozwiązanie problemu producenta – konsumenta za pomocą semaforów.

11. Semafory w standardzie POSIX

Wyróżnione są tu dwa typy semaforów:

1. Semafory nienazwane
2. Semafory nazwane

Semafory nienazwane nadają się do synchronizacji wątków w obrębie jednego procesu. Dostęp do semafora nienazwanego następuje poprzez jego adres. Może on być także użyty do synchronizacji procesów o ile jest umieszczony w pamięci dzielonej.

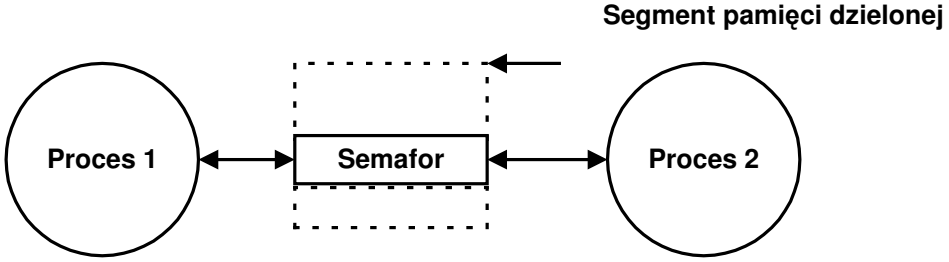
Dostęp do semaforów nazwanych następuje poprzez nazwę. Ten typ semaforów bardziej nadaje się synchronizacji procesów niż wątków. Semafory nienazwane działają szybciej niż nazwane.

sem_open	Utworzenie semafora nazwanego
sem_init	Inicjacja semafora
sem_wait	Czekanie na semaforze
sem_trywait	Pobranie zasobu
sem_timedwait	Czekanie z przeterminowaniem
sem_post	Sygnalizacja
Sem_close	Zamknięcie semafora
Sem_unlink	Kasowanie semafora

Tab. 11-1 Operacje semaforowe w standardzie POSIX

Semafory nienazwane

Dostęp do semafora nienazwanego następuje po adresie semafora. Stąd nazwa semafor nienazwany.



Deklaracja semafora
Przed użyciem semafora musi on być zadeklarowany jako obiekt typu sem_t a pamięć używana przez ten semafor musi zostać mu jawnie przydzielona.

O ile semafor ma być użyty w różnych procesach powinien być umieszczony w wcześniej zaalokowanej pamięci dzielonej.

1. Utworzyć segment pamięci za pomocą funkcji **shm_open**.
2. Określić wymiar segmentu używając funkcji f**trunc**.
3. Odwzorować obszar pamięci wspólnej w przestrzeni danych procesu - **mmap**.

Inicjacja semafora - funkcja sem_init
Przed użyciem semafor powinien być zainicjowany.

```
int sem_init(sem_t *sem, int pshared, unsigned
value)
```

sem Identyfikator semafora (wskaźnik na strukturę w pamięci)
pshared Gdy wartość nie jest zerem semafor może być umieszczony w pamięci dzielonej i dostępny w wielu procesach
value Początkowa wartość semafora

Funkcja zwraca 0 gdy sukces, -1 gdy błąd.

Czekanie na semaforze – funkcja sem_wait
Funkcja ta odpowiada omawianej wcześniej operacji sem_wait(S). Prototyp funkcji jest następujący:

```
int sem_wait(sem_t *sem)
```

sem Identyfikator semafora

Gdy licznik semafora jest nieujemny funkcja zmniejsza go o 1. W przeciwnym przypadku proces bieżący jest zawieszany. Zawieszony proces może być odblokowany przez procedurę sem_sem_post wykonaną w innym procesie lub sygnał. Funkcja zwraca 0 gdy sukces, -1 gdy błąd.

Ograniczone czasowo czekanie na semaforze – funkcja sem_timedwait

Funkcja ta jest wersją operacji sem_wait(S). Prototyp funkcji jest następujący:

```
int sem_timedwait(sem_t *sem, struc timespec timeout)
```

sem Identyfikator semafora
timeout Specyfikacja przeterminowania – czas absolutny

Gdy licznik semafora jest nieujemny funkcja zmniejsza go o 1. W przeciwnym przypadku proces bieżący jest zawieszany.

Zawieszony proces może być odblokowany przez procedurę sem_sem_post wykonaną w innym procesie, sygnał, lub system operacyjny, gdy po upływie czasu timeout zawieszony proces nie zostanie inaczej odblokowany.

Funkcja zwraca 0 gdy sukces, -1 gdy błąd.
Gdy wystąpił timeout kod błędu errno wynosi ETIMEDOUT

```
struct  timespec {
   time_t tv_sec;  // Sekundy
   long   tv_nsec; // nanosekundy
}
```

```
#include <stdio.h>
#include <semaphore.h>
#include <time.h>

main(){
  struct timespec tm;
  sem_t sem;
  int i=0;
  sem_init( &sem, 0, 0);
  do {
        clock_gettime(CLOCK_REALTIME, &tm);
        tm.tv_sec += 1;   i++;
        printf("i=%d\n",i);
        if (i==10) {
            sem_post(&sem);
        }

  } while(sem_timedwait(&sem,&tm) == -1 );
  printf("Semafor zwolniony po %d probach\n", i);
  return;
}
```

Program 11-1 Przykład wykorzystania semafora z przeterminowaniem

Nieblokujące pobranie zasobu

Funkcja ta podobna jest do operacji sem_wait(S). Prototyp funkcji jest następujący:

```
int sem_trywait(sem_t *sem)
```

sem Identyfikator semafora

Gdy licznik semafora jest nieujemny funkcja zmniejsza go o 1. W przeciwnym przypadku funkcja zwraca –1 i proces bieżący nie vjest zawieszany.

Funkcja zwraca 0 gdy sukces, -1 gdy błąd.

Sygnalizacja na semaforze – funkcja sem_post
Funkcja ta odpowiada omawianej wcześniej operacji sem_post(S). Prototyp funkcji jest następujący:

```
int sem_post(sem_t *sem)
```

sem Identyfikator semafora

Jeśli jakikolwiek proces jest zablokowany na tym semaforze przez wykonanie funkcji **sem_wait** zostanie on odblokowany. Gdy brak procesów zablokowanych licznik semafora zwiększany jest o 1. Funkcja zwraca 0 gdy sukces, -1 gdy błąd.

Kasowanie semafora – funkcja sem_destroy
Semafor kasuje się przy pomocy funkcji :

```
int sem_destroy(sem_t *sem)
```

sem Identyfikator semafora

Gdy jakieś procesy są zablokowane na tym semaforze, zostaną one odblokowane i operacja **sem_destroy** zakończy się błędem.

Semafory nazwane

Semafory nazwane identyfikowane są w procesach poprzez ich nazwę.
Na semaforze nazwanym operuje się tak samo jak na semaforze nienazwanym z wyjątkiem funkcji otwarcia i zamknięcia semafora.

Otwarcie semafora – funkcja sem_open
Aby użyć semafora nazwanego należy uzyskać do niego dostęp poprzez funkcję:

```
sem_t *sem_open(const char *sem_name, int oflags,
[int mode, int value])
```

sem_name Nazwa semafora, powinna się zaczynać od znaku „/".
oflags Flagi trybu tworzenia i otwarcia: O_RDONLY, O_RDWR, O_WRONLY. Gdy semafor jest tworzony należy użyć flagi O_CREAT
mode Prawa dostępu do semafora – takie jak do plików. Parametr jest opcjonalny.
value Początkowa wartość semafora. Parametr jest opcjonalny

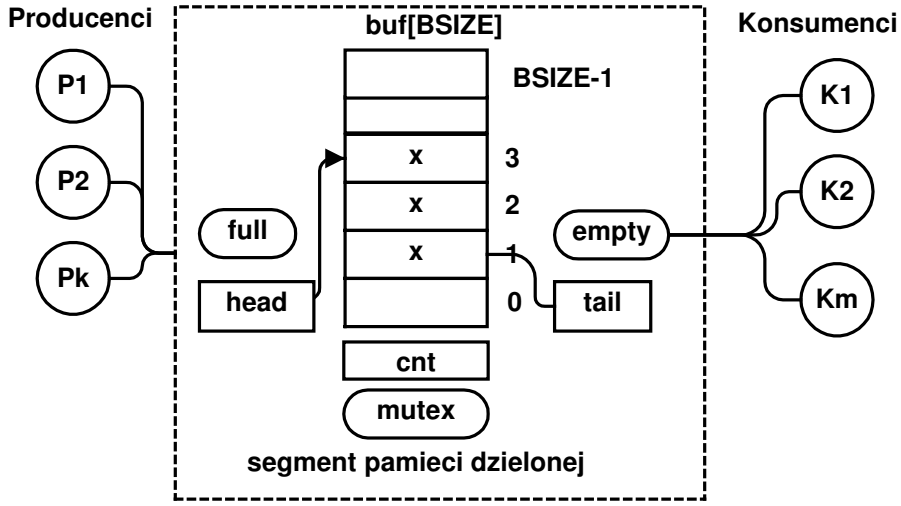
Funkcja zwraca identyfikator semafora.
Semafor widoczny jest w katalogu /dev/sem.
Funkcja tworzy semafor gdy nie był on wcześniej utworzony i otwiera go.

Zamykanie semafora nazwanego – funkcja sem_close
Semafor zamyka się poprzez wykonanie funkcji:

```
int sem_close( sem_t * sem )
```

Funkcja zwalnia zasoby semafora i odblokowuje procesy które są na nim zablokowane. Funkcja zwraca 0 gdy sukces, -1 gdy błąd.

11.1 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");
 fd=shm_open("bufor", O_RDWR|O_CREAT , 0774);
 if(fd == -1){
   perror("open"); exit(-1);
 }

 printf("fd: %d\n",fd);
 size = ftruncate(fd,sizeof(bufor_t));
 if(size < 0) {perror("trunc"); exit(-1); }
 // Odwzorowanie segmentu fd w obszar pamieci procesow
```

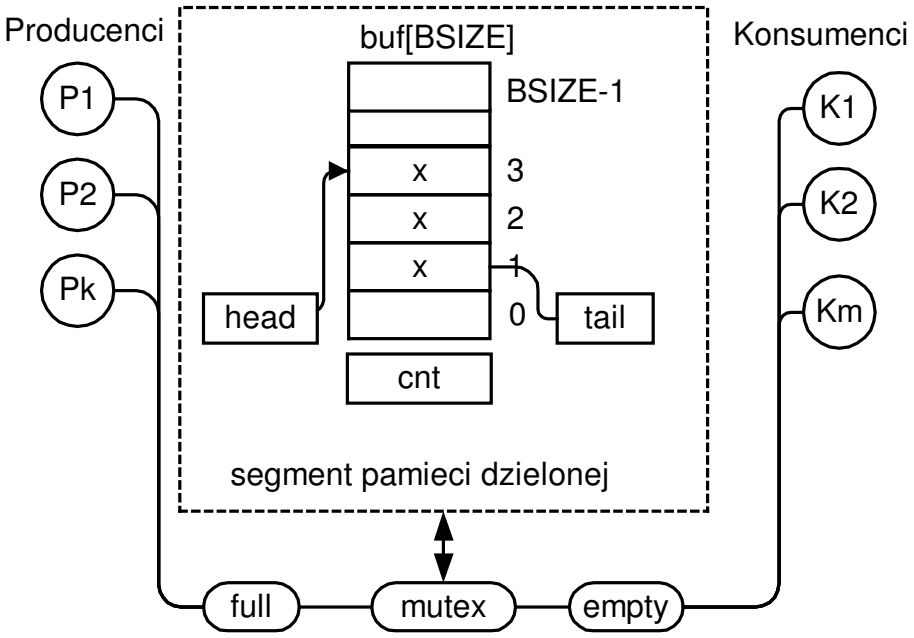
```
wbuf = (bufor_t *)mmap(0, sizeof(bufor_t)
        ,PROT_READ|PROT_WRITE,MAP_SHARED, fd, 0);
if(wbuf == NULL) {perror("map");  exit(-1); }

// 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 11-1 Rozwiązanie problemu producenta i konsumenta za pomocą semaforów nienazwanych

11.2 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 11-2 Rozwiązanie problemu producenta i konsumenta za pomocą semaforów nazwanych