

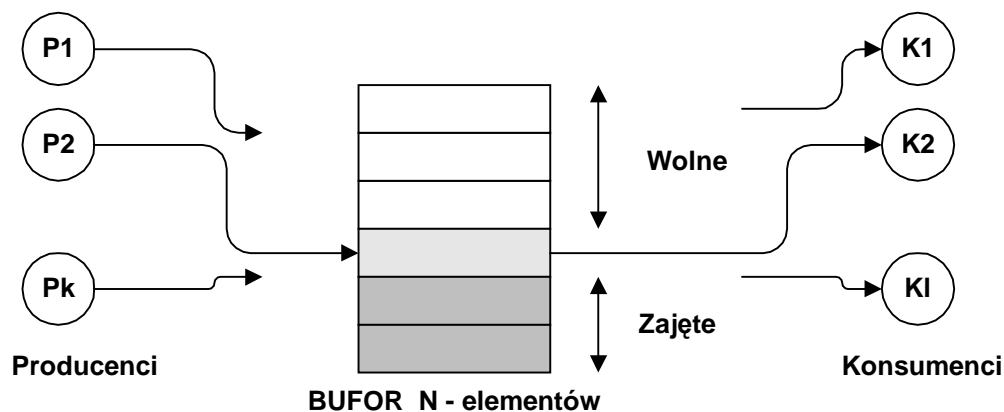
10. Synchronizacja użycia zasobów, Semaforey

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.

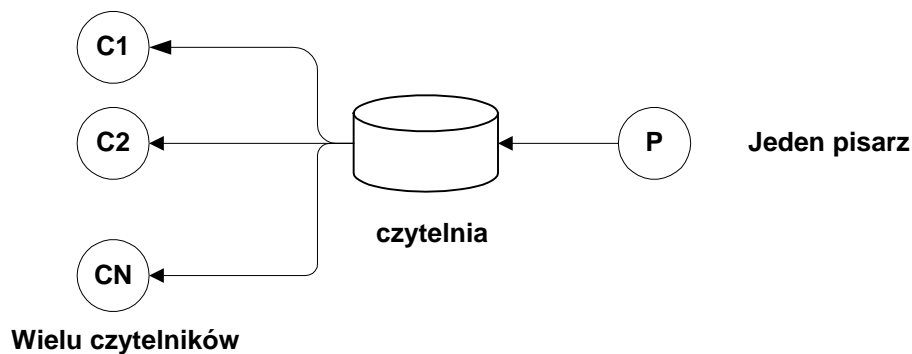
Bufor zorganizowany może być na różnych zasadach.

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

Umieszczanie / pobieranie elementu z bufora jest sekcją krytyczną.

10.2 Problem czytelników i pisarzy

W systemie istnieją dwie grupy procesów – czytelnicy i pisarze. Czytanie może się odbywać współbieżnie z innymi procesami natomiast pisanie, w celu zapewnienia spójności danych, musi się odbywać na zasadzie wyłączości.



- Czytelnik może wejść do czytelni gdy jest ona pusta lub gdy są tam inni czytelnicy
- Pisarz może wejść do czytelni gdy jest ona pusta

Problem jest uogólnieniem problemu dostępu wielu procesów do bazy danych.

10.3 Problem śpiącego fryzjera (sleeping barber problem)

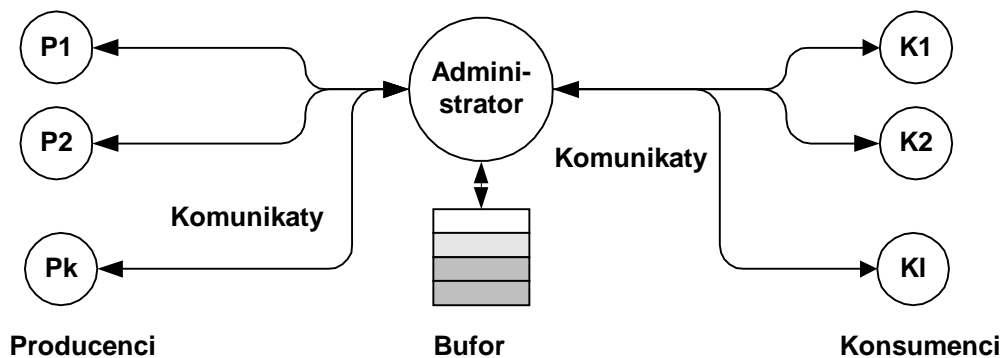
W zakładzie fryzjerskim znajduje się poczekalnia na N miejsc oraz fotel fryzjerski. Gdy przychodzi klient to sprawdza czy są wolne miejsca w poczekalni. Jeżeli tak to zajmuje miejsce, jeżeli nie to przyjdzie on później. Gdy nikogo nie ma i fryzjer śpi to budzi go.

Z kolei fryzjer patrzy czy są jacyś klienci. Gdy tak to bierze na fotel jednego z czekających. Gdy nie to zasypia.

10.4 Rozwiązanie problemu producenta i konsumenta za pomocą komunikatów

Struktura rozwiązania:

1. Zarządzanie buforami leży w gestii procesu administratora
2. Procesy producentów i konsumentów komunikują się z procesem administratora.



Postać komunikatu

```
#define PROD 1 // Producent
#define KONS 2 // Konsument
#define SERV_NAME "/pserver" // Nazwa administratora

typedef struct {
    int type; /* typ procesu */
    int pnr ; /* numer procesu */
    pid_t pid; /* pid procesu */
    char text[SIZE]; /* tekst komunikatu */
} msg_type;
```

Proces producenta i konsumenta

```
main(int argc, char *argv[]) {
    . . .
    // Lokalizacja administratora
    spid = name_open(SERV_NAME, flagi);
    for(i=0; i<10; i++) {
        msg.type = PROD;
        sprintf(msg.text, " %s %d - %d", typ_str, nr, i);
        res =
        MsgSend(spid, &msg, &rmsg, sizeof(msg), sizeof(msg));
        . . .
    }
    exit(0);
}
```

Administrator zasobu

Synchronizacja

Problemy:

- Jak wstrzymać procesy producentów gdy brak miejsca w buforze
- Jak wstrzymać procesy konsumentów gdy brak elementów w buforze

Rozwiązanie:

- Procesom które mają być wstrzymane nie udziela się odpowiedzi Reply.
- Aby można było tak wstrzymany proces potem odblokować należy przechować jego pid.

Struktury danych

```
#define MAX_PROC 8 // Maks. liczba oczekujących procesow
#define BSIZE 2 // Dlugosc bufora komunikatow

// Definicja bufora komunikatow ----
char buffer[BSIZE+1][SIZE]; // Bufor
int bptr; // Indeks

// Bufor zablokowanych procesow producenta
msg_type prod_buf[MAX_PROC]; // Bufor
int prod_ptr; // Liczba czek. producentow

// Bufor zablokowanych procesow konsumenta
msg_type kons_buf[MAX_PROC]; // Bufor
int kons_ptr; // Liczba czek. konsumentow
```

Proces administratora buforów

```
main () {  
  
    // Rejestracja nazwy administratora  
    id = name_attach(NULL,SERV_NAME,flagi);  
    do {  
        pid = MsgReceive(0,&msg,sizeof(msg),NULL);  
        if(msg.type == PROD) { // Producent  
            // Testowanie zajetosci bufora  
            if(bptr < BSIZE ) { // Jest miejsce - kom. do bufora  
                wstaw_do_bufora(&msg);  
                // Mozna zwolnic proces producenta  
                res = MsgReply(pid,0,&msg,sizeof(msg));  
                // Czy jakis konsument czeka na komunikat ?  
                if(kons_ptr > 0) { // Tak czeka  
                    pobierz_z_bufora(&msg);  
                    // Usuń konsumenta z kolejki  
                    kons_ptr--;  
                    pid = kons_buf[kons_ptr].type;  
                    // Odblokuj konsumenta  
                    res = MsgReply(pid,0,&msg,sizeof(msg));  
                }  
            } else { // Brak miejsca w buforze  
                // Zablokuj producenta w kolejce  
                msg.pid = pid;  
                prod_buf[prod_ptr] = msg;  
                prod_ptr++;  
            }  
        }  
        if(msg.type == KONS) { // Konsument  
            if(bptr > 0) { // Jest komunikat - mozna pobrac  
                ....  
                // Czy jakis prod. czeka na miejsce w buforze ?  
                if(prod_ptr > 0) { // Tak czeka  
                    // Zwalniamy producenta  
                    ....  
                }  
            } else { // Brak elementów w buforze  
                // Zablokuj konsumenta w kolejce  
            }  
        }  
    } while(1);  
}
```

Szkic działania administratora buforów

10.5 Semafor

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

Definicja semafora.

Semafor S jest obiektem abstrakcyjnym 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	<code>sem_init(S,N)</code>	Ustawienie licznika semafora S na początkową wartość N .
Zajmowanie	<code>sem_wait(S)</code>	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	<code>sem_post(S)</code>	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: $L \geq 0$
2. Jego wartość wynosi: $L = N - \text{Liczba_operacji}(\text{sem_wait}) + \text{Liczba_operacji}(\text{sem_post})$. (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.

10.6 Zastosowania semaforów

Implementacja wzajemnego wykluczenia

```

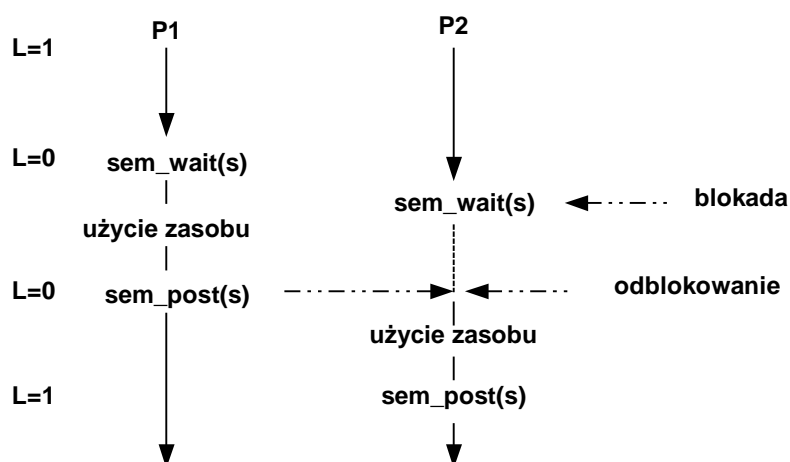
semaphore S; // Deklaracja semafora
sem_init(S,1); // Inicjacja semafora

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

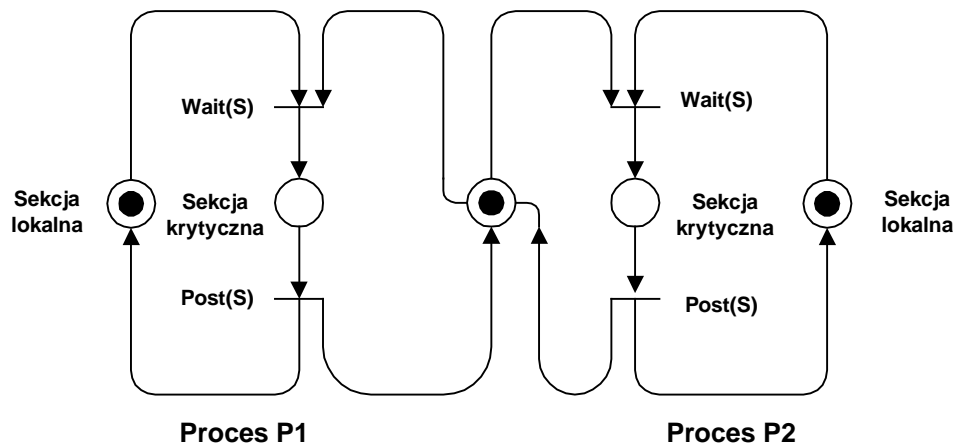
Proces2 {
do {
    Sekcja_lokalna;
    sem_wait(S);
    Sekcja_krytyczna;
    sem_post(S);
} while(1);
}

```

Program 10-1 Przykład zastosowania semafora do ochrony sekcji krytycznej



Rys. 10-1 Przebieg operacji blokowania



Rys. 10-2 Implementacja wzajemnego wykluczenia za pomocą semafora – Ilustracja za pomocą sieci Petriego

Rozwiązanie problemu producenta – konsumenta

```

#define BufSize 8 // Bufor ma 8 elementów
RecType Buffer[BufSize]; // Bufor na elementy
semaphore Mutex; // Ochrona bufora
semaphore Puste; // Wolne bufory
semaphore 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; // Czekać na wolny bufor
    // Czekać 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; x = Buffer[count];
    count++; count--;
    sem_post(Mutex); sem_post(Mutex);
    // Pojawił się nowy element // Zwolnij miejsce w buforze
    sem_post(Pelne); konsumpcja rekordu x;
  } while(1); ...
} } while(1);
} }

main(void) {
  count = 0; // Inicjacja semafora Puste
  sem_init(Puste, BufSize); // Inicjacja semafora Pelne
  sem_init(Pelne, 0); // Inicjacja semafora Mutex
  sem_init(Mutex, 1); // Start K wątków producenta
  StartThread(producent, ..);
  ..
  StartThread(konsument, ..); // Start L wątków konsumenta
  ..
}

```

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

10.7 Rozwiązanie problem czytelników i pisarzy za pomocą semaforów

Rozwiązanie z możliwością zagłodzenia pisarzy

- Czytelnik może wejść do czytelnicy gdy jest ona pusta lub gdy są tam inni czytelnicy
- Pisarz może wejść do czytelnicy gdy jest ona pusta

Może się tak zdarzyć że zawsze jakiś czytelnik jest w czytelnicy co doprowadzi do zagłodzenia pisarzy.

```
semaphore mutex ; // Kontrola dostępu do reader_count
semaphore db; // Kontrola dostępu do czytelnicy
int reader_count; // Liczba czytelników w czytelnicy

Reader(){
    while (TRUE) {
        sem_wait(mutex); // Blokada dostępu do reader_count
        reader_count = reader_count + 1;

        // Pierwszy czytelnik blokuje dostęp do czytelnicy pisarzom
        if (reader_count == 1)
            sem_wait(db);
        // Zwolnienie dostępu do reader_count
        sem_post(mutex);
        read_db(); // Czytelnik czyta
        sem_wait(mutex); // Blokada dostępu do reader_count
        reader_count = reader_count - 1;

        // Gdy nie ma innych czytelników to wpuszczamy pisarzy
        if (reader_count == 0)
            sem_post(db);
        sem_post(mutex); // Zwolnienie dostępu do reader_count
    }
}

Writer() {
    while (TRUE) {
        create_data(); // Pisarz zastanawia się
        sem_wait(db); // Pisarz czeka na dostęp do czytelnicy
        write_db(); // Pisarz w czytelnicy - pisze
        sem_post(db); // Pisarz zwalnia dostęp do czytelnicy
    }
}
```

```
main() {  
    sem_init(mutex,1);  
    sem_init(db,1);  
    ....  
}
```

Program 10-3 Problem czytelników i pisarzy – rozwiązanie za pomocą semaforów

Rozwiązanie z możliwością zagłodzenia czytelników

- Czytelnik musi czekać gdy są w czytelni lub czekają jacykolwiek pisarze

Rozwiązanie poprawne

- Wpuszczać na przemian czytelników i pisarzy
- Gdy wchodzi jeden z czytelników, to wpuszcza on wszystkich czekających czytelników

Rozwiązanie poprawne nie dopuszcza do zagłodzenia czy to czytelników czy też pisarzy.

```
#define PLACES 8           // Liczba wolnych miejsc w czytelni
semaphore wolne ;        // Liczba wolnych miejsc w czytelni
semaphore wr;           // Kontrola dostępu do czytelni

Reader()
{
    while (TRUE) {
        sem_wait(wolne);    // Czekanie na miejsca w czytelni
        read_db();          // Czytelnik w czytelni - czyta
        sem_post(wolne);    // Zwolnienie miejsca w czytelni
    }
}

Writer() {
    while (TRUE) {
        create_data();      // Pisarz zastanawia się
        sem_wait(wr);       // Zablokowanie dostępu dla pisarzy
        // Wypieranie czytelników z czytelni
        for(j=1;j<=places;j++)
            sem_wait(wolne);
        write_db();         // Pisarz w czytelni – pisze
        // Wpuszczenie czytelników
        for(j=1;j<= PLACES;j++)
            sem_post(wolne);
        sem_post(wr);       // Odblokowanie pisarzy
    }
}

main() {
    sem_init(wolne,PLACES);
    sem_init(wr,1);
    ....
}
```

Program 10-4 Rozwiązanie z ograniczoną liczbą miejsc w czytelni

11. Semaforey w standardzie POSIX

Wyróżnione są tu dwa typy semaforów:

1. Semaforey nienazwane
2. Semaforey nazwane

Semaforey 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.

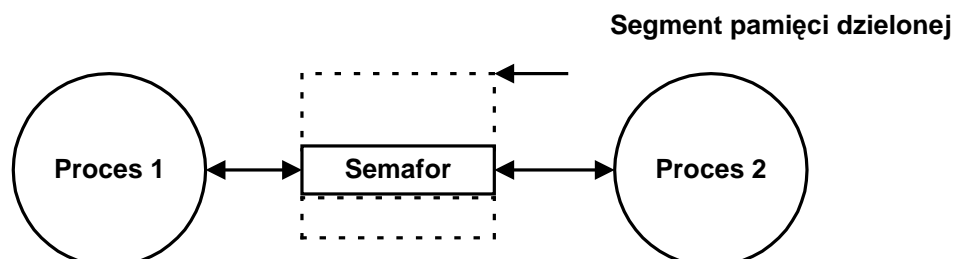
Dostęp do semaforów nazwanych następuje poprzez nazwę. Ten typ semaforów bardziej nadaje się synchronizacji procesów niż wątków. Semaforey nienazwane działają szybciej niż nazwane.

<code>sem_open</code>	Utworzenie semafora nazwanego
<code>sem_init</code>	Inicjacja semafora
<code>sem_wait</code>	Czekanie na semaforze
<code>sem_trywait</code>	Pobranie zasobu
<code>sem_timedwait</code>	Czekanie z przeterminowaniem
<code>sem_post</code>	Sygnalizacja
<code>sem_close</code>	Zamknięcie semafora
<code>sem_unlink</code>	Kasowanie semafora

Tab. 11-1 Operacje semaforowe w standardzie POSIX

Semaforey 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 `ltruncate`.
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_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, struct 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_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 jest 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.

Semafor nazwany

Semafor nazwany identyfikowany jest w procesach poprzez jego 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.

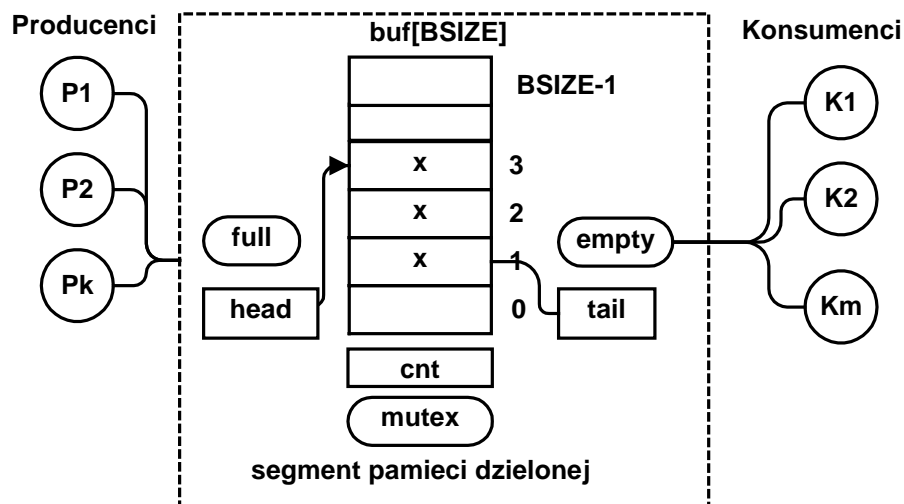
Zamykanie semafora nazwanego – funkcja sem_close

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");
    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 11-1 Rozwiązanie problemu producenta i konsumenta za pomocą semaforów nienazwanych