

# Problem producenta i konsumenta

Mateusz Pater

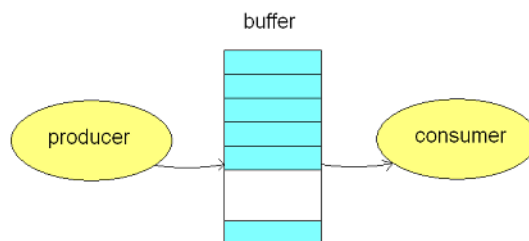
## Spis treści

1	Opis problemu	1
2	Rozwiązanie	2
3	Implementacja	2
4	Testowanie	4

## 1 Opis problemu

Problem producenta i konsumenta to klasyczny informatyczny problem synchronizacji. W problemie występują dwa rodzaje procesów: producent i konsument, którzy dzielą wspólny zasób - bufor dla produkowanych (konsumowanych) jednostek. Zadaniem producenta jest wytworzenie produktu, umieszczenie go w buforze i rozpoczęcie pracy od nowa. W tym samym czasie konsument ma pobrać produkt z bufora. Problemem jest taka synchronizacja procesów, żeby producent nie dodawał nowych jednostek gdy bufor jest pełny, a konsument nie pobierał gdy bufor jest pusty.

Rozwiązaniem dla producenta jest uśpienie procesu w momencie gdy bufor jest pełny. Pierwszy konsument, który pobierze element z bufora budzi proces producenta, który uzupełnia bufor. W analogiczny sposób usypiany jest konsument próbujący pobrać z pustego bufora. Pierwszy producent, po dodaniu nowego produktu umożliwi dalsze działanie konsumentowi. Rozwiązanie wykorzystuje komunikację międzyprocesową z użyciem semaforów. Nieprawidłowe rozwiązanie może skutkować zakleszczeniem. [1]



Rysunek 1: Problem producenta i konsumenta

## 2 Rozwiązanie

W rozwiązaniu poniżej używamy dwóch semaforów: pusty oraz pełny. Semafor pusty jest opuszczany przed dodaniem do bufora. Jeśli bufor jest pełny semafor nie może być opuszczony i producent zatrzymuje się przed dodaniem. W następnym uruchomieniu konsumenta semafor jest podniesiony, co umożliwia producentowi dodanie jednostki do bufora. Konsument działa w analogiczny sposób.

semaphore pelny = 0  
semaphore pusty = rozmiar bufora

```

procedure producent
while true do
    produkt = produkuj();
    down(pusty);
    dodajProduktDoBufora(produkt);
    up(pelny);
end

procedure konsument
while true do
    down(pelny);
    produkt = pobierzProduktZBufora() ;
    up(pusty) uzyjProdukt(produkt)
end

```

Powyższe rozwiązanie działa poprawnie w przypadku, gdy istnieje tylko jeden producent i tylko jeden konsument. W czasie działania wielu procesów może dojść do próby jednoczesnego odczytania bądź zapisania produktu w buforze w tym samym miejscu.

## 3 Implementacja

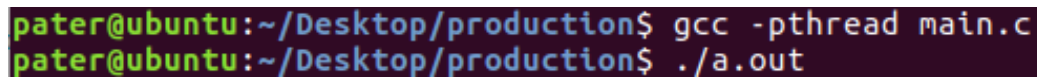
Implementacja problemu w języku C w której korzystamy z **mutex** w celu zabezpieczenia sekcji krytycznej, którą u nas jest działanie na zmiennej **count**).

### Mutex:

Is a key to a toilet. One person can have the key - occupy the toilet - at the time. When finished, the person gives (frees) the key to the next person in the queue.

### Semaphore:

Is the number of free identical toilet keys. Example, say we have four toilets with identical locks and keys. The semaphore count - the count of keys - is set to 4 at beginning (all four toilets are free), then the count value is decremented as people are coming in. If all toilets are full, ie. there are no free keys left, the semaphore count is 0. Now, when eq. one person leaves the toilet, semaphore is increased to 1 (one free key), and given to the next person in the queue.



```
pater@ubuntu:~/Desktop/production$ gcc -pthread main.c
pater@ubuntu:~/Desktop/production$ ./a.out
```

Rysunek 2: Kompilacja programu przez terminal

Inicjalizacja zmiennych i wątków

```
pthread_mutex_t mutex;
sem_t isEmpty, isFull;
int count = 0;

pthread_mutex_init(&mutex, NULL);
sem_init(&isEmpty, 0, 10); //10 jest to rozmiar naszego bufora
sem_init(&isFull, 0, 0);

pthread_t threads[2];
pthread_create(&threads[0], NULL, producer, NULL);
pthread_create(&threads[1], NULL, consumer, NULL);
pthread_join(threads[0], NULL);
pthread_join(threads[1], NULL);
```

Konsument

```
void *consumer() {
    while (1) {
        sem_wait(&isFull);
        pthread_mutex_lock(&mutex);
        --count;
        pthread_mutex_unlock(&mutex);
        sem_post(&isEmpty); //zwolnij magazyn
    }
}
```

Producent

```
void *producer() {
    while (1) {
        sem_wait(&isEmpty); //czekaj na zwolnienie magazynu
        pthread_mutex_lock(&mutex);
        ++count;
        pthread_mutex_unlock(&mutex);
        sem_post(&isFull);
    }
}
```

## 4 Testowanie

Musimy przetestować teraz naszego konsumenta oraz producenta. Konsument, według założeń, nie ma prawa wziąć nieistniejącego produktu co jest równoważne z tym, że nasza zmienna **count** nigdy nie będzie ujemna. Uśpijmy wątek producenta na pewien kwant czasu oraz zmieńmy delikatnie implementację i użyjmy metody **printf**, aby sprawdzić co się dzieje podczas każdego obiegu pętli.

```
void *producer() {
    while (1) {
        sem_wait(&isEmpty);
        pthread_mutex_lock(&mutex);
        printf("Wyprodukowałem, aktualna wielkosc magazynu: %d\n", ++count);
        pthread_mutex_unlock(&mutex);
        sem_post(&isFull);
        sleep(2);
    }
}
```

```
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
Wyprodukowałem, aktualna wielkosc magazynu: 1
Pobrałem, aktualna wielkosc magazynu: 0
```

Mimo, że wątek producenta został uśpiony to konsument i tak czekał z pobraniem na zapelnienie bufora.

Analogiczna sytuację mamy w przypadku naszego producenta. Nie powinien on produkować nowych przedmiotów jeśli bufor jest zapelniony. Uśpijmy w takim razie konsumenta i upewnijmy się, że tak jest.

```
void *consumer() {
    while (1) {
        sem_wait(&isFull);
        pthread_mutex_lock(&mutex);
        printf("Pobrałem, aktualna wielkosc magazynu: %d\n", --count);
        pthread_mutex_unlock(&mutex);
        sem_post(&isEmpty);
        sleep(2);
    }
}
```

```
Wyprodukowałem, aktualna wielkosc magazynu: 4
Pobrałem, aktualna wielkosc magazynu: 3
Wyprodukowałem, aktualna wielkosc magazynu: 4
Wyprodukowałem, aktualna wielkosc magazynu: 5
Pobrałem, aktualna wielkosc magazynu: 4
Wyprodukowałem, aktualna wielkosc magazynu: 5
Wyprodukowałem, aktualna wielkosc magazynu: 6
Pobrałem, aktualna wielkosc magazynu: 5
Wyprodukowałem, aktualna wielkosc magazynu: 6
Wyprodukowałem, aktualna wielkosc magazynu: 7
Pobrałem, aktualna wielkosc magazynu: 6
Wyprodukowałem, aktualna wielkosc magazynu: 7
Wyprodukowałem, aktualna wielkosc magazynu: 8
```

## **Literatura**

- [1] Problem producenta i konsumenta  
[https://pl.wikipedia.org/wiki/Problem\\_producenta\\_i\\_konsumenta](https://pl.wikipedia.org/wiki/Problem_producenta_i_konsumenta)