Mateusz Pater

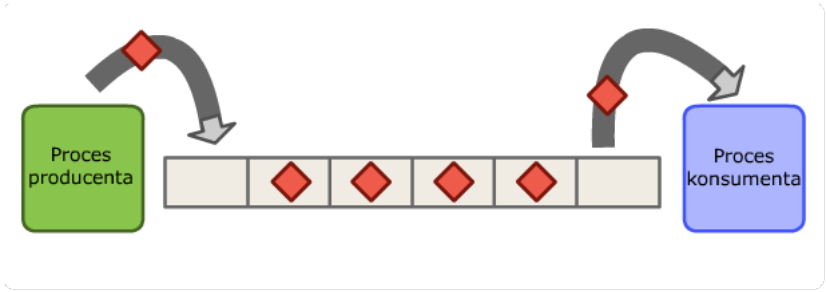
Problem producenta i konsumenta

Spis treści

1. Opis problemu

Problem producenta i konsumenta to klasyczny informatyczny problem synchronizacji. W problemie występują dwa rodzaje procesów: producent i konsument, które dzielą wspólny zasób - bufor na produkowane (konsumowanych) jednostki. 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 dotyczącym procesu producenta jest uniemożliwienie mu opuszczenia semaforu 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. Wadliwe rozwiązanie może skutkować zakleszczeniem [1].



Rysunek 1. Zobrazowanie problemu producenta i konsumenta [2]

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 pełny = 0
semaphore pusty = rozmiar bufora
procedure (producent)
while true do
    produkt = produkuj();
    down(pusty);
    dodajProduktDoBufora(produkt);
    up(pełny);
end
```

Algorithm 1: Procedura producenta

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

Algorithm 2: Procedura konsumenta

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

3. Implementacja

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

Główna struktura, która zawiera dwa semafory oraz zmienną **licznik**:

```
struct Shared{
    sem_t isEmpty ;
    sem_t isFull ;
    int licznik ;
} * g_shm ;
```

Tworzenie wspólnej struktury g_shm, zapoczątkowanie semaforów oraz ustawienie początkowej wartości zmiennej licznik:

```
g_shm = mmap( NULL , sizeof( struct Shared ) ,PROT_READ | PROT_WRITE , MAP_SHARED |
MAP_ANONYMOUS , -1 , 0 );
sem_init( & g_shm->isEmpty , 1 , 10 );
sem_init( & g_shm->isFull , 1 , 0 );
g_shm->licznik = 0 ;
```

Utworzenie procesu konsumenta oraz uruchomienie producenta:

```
    int id = 0;
    id = fork();
    if (id < 0)
    {
perror ("error\n");
exit(1);
    }
    else if (id == 0)
    {
          consumer();
          exit(0);
    }

        producer();
```

Producent:

```
void producer(void) {
    while (1) {
        sem_wait(&g_shm->isEmpty);
++g_shm->licznik;
        sem_post(&g_shm->isFull);
    }
 }
```

Konsument:

```
void consumer(void) {
    while (1) {
        sem_wait(&g_shm->isFull);
--g_shm->licznik;
        sem_post(&g_shm->isEmpty);
    }
}
```

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 **licznik** nigdy nie będzie ujemna. Uśpijmy wątek producenta na pewien czas oraz zmieńmy nieco implementację i użyjmy funkcji `printf`, aby sprawdzić co się dzieje podczas każdego obiegu pętli.

```
void producer(void) {
    while (1) {
        sem_wait(&g_shm->isEmpty);
        printf("Wyprodukowalem, aktualna wielkosc magazynu: %d\n", ++g_shm->count);
        sem_post(&g_shm->isFull);
        sleep(1);
    }
}
```

Rysunek 2. Uśpienie producenta na sekundę metodą sleep

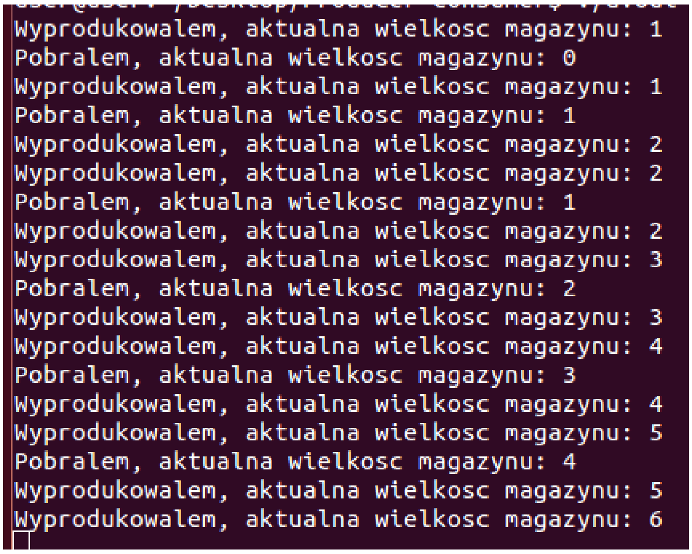
```
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
Wyprodukowalem, aktualna wielkosc magazynu: 1
Pobralem, aktualna wielkosc magazynu: 0
```

Rysunek 3. Reakcja konsumenta na uśpienie producenta

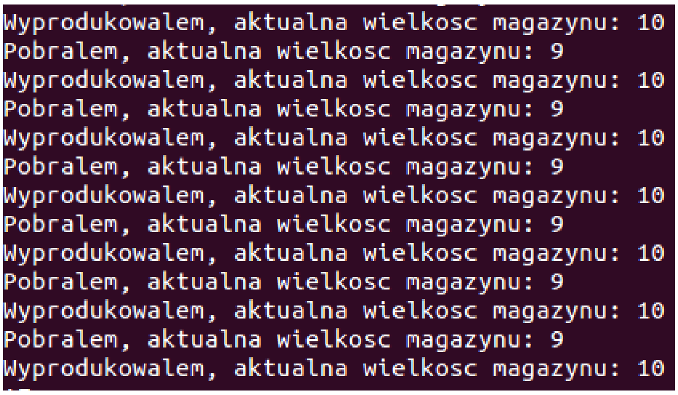
Mimo że wątek producenta został uśpiony, to konsument i tak czekał z pobraniem na zapełnienie bufora (zob. rys. 3). Analogiczna sytuację mamy w przypadku producenta. Nie powinien on produkować nowych przedmiotów, jeśli bufor jest zapełniony. Uśpijmy w takim razie konsumenta i upewnijmy się, że tak jest (zob. rys. 5).

```
void consumer(void) {
    while (1) {
        sem_wait(&g_shm->isFull);
        printf("Pobralem, aktualna wielkosc magazynu: %d\n", --g_shm->count);
        sem_post(&g_shm->isEmpty);
        sleep(1);
    }
}
```

Rysunek 4. Uśpienie konsumenta na sekundę funkcją `sleep`



Rysunek 5. Reakcja producenta na uśpienie konsumenta



Rysunek 6. Zapełniony bufor, producent nie wyprodukuje więcej niż 10 produktów

Literatura

1. Problem producenta i konsumenta *https://pl.wikipedia.org/wiki/Problem_producenta_i_konsumenta*

2. O procesach możemy poczytać pod adresem *http://iair.mchtr.pw.edu.pl/ pwnuk/podreczniki/podrecznik_syst_komputerowe/lekcja6/segment2*