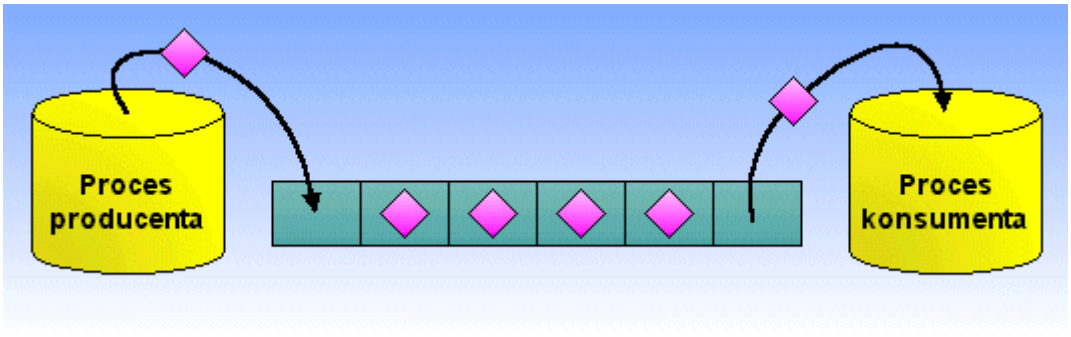
Szymon Kopa

Problem producenta i konsumenta
2018-02-02

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 (konsumowane) 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



Zobrazowanie problemu
Źródło: www.isep.pw.edu.pl

2. Rozwiązanie problemu

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.

```
semafor pelny = 0
semafor pusty = rozmiar_bufora

procedure producent(){
   while true{
      produkt = produkuj();
      down(pusty);
      dodajProduktDoBufora(produkt);
      up(pelny);
   }
}

procedure konsument(){
   while true{
      down(pelny);
      produkt = pobierzProduktZBufora();
      up(pusty);
      uzyjProdukt(produkt);
   }
}
```

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)
{
   perrod(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. Uruchomienie

Aby uruchomić program wystarczy w terminalu przejść do katalogu, w którym znajduje się plik z kodem programu (so.c) i wpisać polecenie:

```
gcc -o so so.c -pthread
```

następnie:

```
./so
```