Na czym polegają błędy synchronizacji procesów związane z korzystaniem z pamięci współdzielonej?

a) problem ograniczonego buforowania :

problem polega na tym ze mamy ograniczoną liczbę zasobów i procesy nie mogą w nieskończoność zapisać do bufora, bo nastąpi przepełnienie. Dlatego operujemy ograniczoną liczbą buforów ,do jednego będzie proces pierwszy pisać i z tego samego będzie drugi proces czytać po podaniu zgody od procesu zapisującego . w przykładzie procesu producenta i konsumenta mamy n buforów

wzajwy =1 służy do spełniania warunku wzajemnego wyłączania
pusty =n mam n pustych buforów
pełny =0 nie ma żadnych pełnych buforów

operacje czekaj i sygnalizuj służą do udostępnienie dostępu i zabronienie go procesom

czekaj(semafor) : while s<=0 do nic;
s:=s-1;

sygnalizuj(semafor) : s:=s+1;

operacje te służą do tego żeby gdy jeden proces modyfikuje zawartość semafora żaden inny proces nie mógł jednocześnie je zmienić

proces producenta :

Produkuj jednostki do producent
czekaj(pusty) mamy niej pustych buforów o jeden
czekaj(wzajwy) wartość semafora jest równa 0, zabrania innym procesom dostęp do bufora w czasie dodawania jednostek do bufora ponieważ konsument tez może zgłosić chęć konsumowania z bufora

Dodaj jednostki z producent do bufora
sygnalizuj(wzajwy) pozwala konsumentowi konsumować z bufora ponieważ wartość semafora=1
sygnalizuj(pełny) semafor pełny się zwiększa o jeden bo mamy mniej pustych o jeden buforów

proces konsumenta :
czekaj(pełny) jest mniej o jeden pełnych buforów bo konsumujemy wyprodukowane zasoby
czekaj(wzajwy) zabrania producentom zapisać do bufora ponieważ wzajwy=0

wyjmowanie jednostki z bufora do konsument
sygnalizuj(wzajwy) pozwala innym procesom pracować i zapisać do bufora nowe jedostki
sygnalizuj(pusty) ma jeden dodatkowy pusty bufor bo zostały skonsumowany jednostki w nim

konsumuj z konsument

algorytm ograniczonego buforowania służy do rozwiązania problemu synchronizacji procesów

b)problem czytelników i pisarzy:

zasób może być dzielony między kilkoma procesami ,niektóre z tych procesów będą czytać zawartość tego zasobu ,inne procesy będą zarówno czytać jak i pisać ,czytelnicy maja jednoczesny dostęp do obiektu ,zaś pisarzy nie mogą mieć jednoczesnego dostępu do obiektu danych. Dlatego czekają w kolejce ,czytelnik może czekać gdy przed nim proces pisarza czeka w kolejce. Rozwiązaniem tego problemu jest algorytm następujący

var wzajwy ,pisarzy: semafor
licznik: integer
wzajwy =1 *służy do spełnienia warunku wzajemnego wyłaczania*
pisarzy=1 *jest dla wszystkich procesów czytelników i pisarzy służy do gwarantowania wzajemnej wyłączności pracy pisarzy*
licznik=0 *jest to liczba procesów czytających obiektu*

struktura procesu pisarza:
czekaj(pisarz)
pisanie….
Sygnalizuj(pisarz)

Jeżeli na pisarze czeka n czytelników , to jeden czytelnik stoi w kolejce do semafora *pisarz* , n-1 czytelników ustawia się do bufor *wzajwy*. *Gdy pisarz dokonuje sygnalizuj(pisarz) to może wznowić pracę czekających czytelników ,albo jednego pisarza .*

Struktura procesu czytelnika:
Czekaj(wzajwy) *zabrania czytać innym czytelnikom*
Licznik+=1
If licznik=1 then czekaj(pisarz) *zabronienie pisarzom pisać*

Sygnalizuj(wzajwy) *czytelnicy mogą czytać obiekt danych*

CZYTANIA….
Czekaj(wzajwy) *koniec czytania*
Licznik=licznik-1
If licznik=0 then sygnalizuj(pisarz) *dawanie pisarzom praw pisania*
Sygnalizuj(wzajwy) *czytelnicy mogą kandydować się on dostęp do obiektu*

b) problem posilających się filozofów:

mamy pięć filozofów na stole z pięcioma pałeczkami do jedzenia , filozof rozpocznie jedzenie gdy podniesie dwie pałeczki ,wtedy sąsiedni filozof nie będzie mógł zacząć jedzenia .problem polega na skonstruowanie algorytmu rozwiązującego problemu synchronizacji

```
var paleczka: array[0..4]of semafor;
repeat
    czekaj(paleczka[i]);
   czekaj(paleczka[i + 1 mod 5]);
   jedzenie
  sygnalizuj(paleczka[i]);
  sygnalizuj(paleczka[i + 1 mod 5]);
until false
```

to rozwiązanie z tym że zagwarantuje że w żadnej chwili nie będą jedli jednocześnie ,ale można w nim można wykryć możliwość powstania blokady gdy pięć filozofów razem będą chcieli podnieść pałeczki razem . jest kilka możliwości unikania tej blokady:

1)pozwolić filozofowi podnieść pałeczkę gdy oba (lewa i prawa) są wolne

2)zastosować rozwiązanie asymetryczne że, filozof o numerze nieparzystym może podnieść najpierw pałeczkę po lewej stronie ,parzysty pałeczkę po prawej stronie .

Co to jest sekcja krytyczna?

Każdy proces ma segment kodu zwany sekcja krytyczna , w której może zmienić wspólne zmienne . Gdy jeden proces wykonuje swoją sekcję krytyczną to żaden inny proces nie może wejść do swojej sekcji krytycznej. Problem sekcji krytycznej polega na skonstruowanie protokołu organizającego współpracy procesów .

Ogólna struktura wygląda następująco :

```
Repeat
        Sekcja wejściowa
          Sekcja krytyczna
        Sekcja wyjściowa
            Reszta          // jest to pozostały kod
Until false;
```

Warunki prawidłowego rozwiązania problemu sekcji krytycznej?

Rozwiązanie problemu sekcji krytycznej musi spełnić trzy warunki :

- **Wzajemne wykluczanie: gdy jeden proces działa w sekcji krytycznej to żaden inny proces nie ma dostęp do swojej sekcji krytycznej**
- **Postęp: jeżeli nie ma żadnych procesów w sekcji krytycznej i inne procesy chcą wejść do sekcji krytycznej to mogą kandydować procesy nie wykonujących swoich reszt jako pierwsze**
- **Ograniczone czekanie: musi istnieć wartość graniczna liczby wejść innych procesów do ich sekcji krytycznej po tem , gdy dany proces zgłosił chęć wejścia i zanim uzyskał na to pozwolenia**

Co to jest semafor?

Jest to narzędzie synchronizacji ,który znajduje zastosowanie w rozwiązaniu problemu sekcji krytycznej z udziałem n procesów . semafor jest dostępny tylko za pomocą operacji czekaj i sygnalizuj

Czekaj(wzajwy) wzajwy =(wzajwy=1)-1=0 nie można wejść do sekcji krytycznej
Sekcja krytyczna tylko jeden proces jest w sekcji krytycznej
Sygnalizuj(wzajwy) wzajwy jest ustawiony na 1 może następny proces wejść

Semafory mogą służyć do rozwiązania problemu synchronizacji np. dla operacji p1(s1), p2(s2)

S1;

Sygnalizuj(synch); **po wykonaniu s1 może instrukcja s2 się wykonać synch=1**
Czekaj(synch) ; **tylko ta instrukcja na razie się wykonuje**
S2;

Opracowanie : **Bnayat Wasim** W.Bnayat.2@wsisiz.edu.pl