Punkty:

- Program współbieżny
- Wzajemne wykluczanie
- Poprawność
- Poziom abstrakcji
- Instrukcje pierwotne, instrukcje atomowe
- Abstrakcja programowania współbieżnego
- Wzajemne wykluczanie przy użyciu semaforów

Program współbieżny

Program współbieżny - kilka sekwencyjnych procesów, których wykonania mogą być przeplatane

- procesy niezależne
- bez uwarunkowań czasowych

Środki komunikacyjne:

- wspólna pamięć – pseudo-równoległe systemy z przełączaniem
- wspólne procedury
- wspólne współdzielone dane

Wzajemne wykluczanie

- Fragment A1 procesu P1 oraz fragment A2 procesu P2 wykluczają się nawzajem, jeśli wykonanie A1 nie może nałożyć się w czasie z wykonaniem fragmentu A2.

- Jeśli procesy P1 i P2 równocześnie starają się wykonać odpowiednio fragmenty A1 i A2, to musi być zapewnione, że tylko jednemu z nich się to uda.

- Przegrywający proces musi być zawieszony, to znaczy musi nastąpić wstrzymanie jego działania aż do zakończenia wykonywania właściwego fragmentu programu przez proces który wygrał.

Typowe sytuacje:

- Przydział zasobów
- Rozwiązanie wymaga zawieszenia współbieżności

Wzajemne wykluczanie:

- *Strefa lokalna*
- *Protokół wstępny*
- *Strefa krytyczna*
- *Protokół końcowy*

Strefa lokalna	–	przetwarzanie prywatne
Protokół wstępny	–	jak wejść do sekcji krytycznej
Strefa krytyczna	–	tylko jeden proces w tej sekcji
Protokół końcowy	–	wyjście ze strefy krytycznej, udostępnienie sekcji krytycznej innym

Protokoły – dodatkowe koszty związane ze współbieżnością

Poprawność:

- Powinien wykonywać to, o czym mówią jego specyfikacje (truizm)
- Bezpieczeństwo (safety)
- Żywotność (liveness)

Globalny brak żywotności

- Blokada (deadlock)
- Globalne zatrzymanie w oczekiwaniu
- Efekt całkowity – zatrzymanie pracy

Lokalny brak żywotności

- Wykluczenie (lockout)
- Zagłodzenie (starvation)
- Można zidentyfikować proces który będzie zawieszony w nieskończoność
- Trudne do wykrycia
- Zdarza się w określonych sytuacjach
- Mniej groźne niż blokada

- Uczciwość (fairness) – sensowne prawa dostępu do sekcji krytycznej

Poziom abstrakcji

- Przesunięcie problemu na niższy poziom sprzętowy
- Dostęp do pojedynczego słowa pamięci:
- Chcemy wykonać: n=n+1

1. Załaduj n
2. Dodaj 1
3. Zapisz n

Scenariusz:

- N=6
- Proces P1 wykonuje ‘załaduj N’
- Zwiększa N
- Proces P2 wykonuje ‘załaduj N=6’
- Zwiększa N
- P1 zapisuje N
- P2 zapisuje N

Instrukcje pierwotne, instrukcje atomowe:

- Niepodzielne:
- Pierwotna instrukcja: wykonanie krytycznej procedury
- Dwa procesy chcą wykonać to tylko jednemu się uda,
- Kolejny problem – przerwania.

Abstrakcja programowania współbieżnego:

- Program współbieżny składa się z dwóch lub więcej programów sekwencyjnych, których wykonania są przeplatane

- Procesy są luźno połączone, błąd w jednym z procesów poza strefą krytyczną i protokołami nie ma wpływu na inne procesy

- Program współbieżny jest poprawny, jeśli respektuje własności bezpieczeństwa, takie jak wzajemne wykluczanie, oraz własności żywotności, takie jak brak blokad i wykluczeń

- Program współbieżny jest niepoprawny, jeśli istnieje przeplot wykonań, który nie spełnia wymagania poprawności. Wystarczy przedstawić jeden scenariusz, aby wykazać niepoprawność. Wykazanie poprawności wymaga przedstawienia matematycznego argumentu uzasadniającego, że program jest poprawny dla wszystkich możliwych wykonań

- Nie czyni się żadnych założeń na temat zależności czasowych, poza tym, że żaden z procesów nie zatrzymuje się wewnątrz strefy krytycznej, oraz że spośród kilku gotowych do działania procesów przynajmniej jeden podejmie działanie.

- W języku programowania występują pierwotne instrukcje synchronizacyjne. Nie wchodzimy w istotę ich implementacji. Zdefiniowane jest tylko ich składnia i semantyka.

Podstawowy problem:

Każdy z dwóch procesów P1 i P2 wykonuje się w nieskończonej pętli składającej się odpowiednio z dwóch części: strefy krytycznej (odpowiednio kryt1 i kryt2) oraz strefy lokalnej (lok1 i lok2) stanowiącej resztę programu. Wykonanie kryt1 i kryt2 nie może odbywać się równocześnie.

Rozwiązanie:

Tablica z zapisem, który proces ma wyjść do sekcji krytycznej:
Załóżmy, że inicjalnie 1:

P1:
Sprawdza czy na tablicy jest 1
Wykonuje sekcję krytyczną
Zapisuje na tablicy 2

P2:
Sprawdza, czy na tablicy jest 2
Wykonuje sekcję krytyczną
Zapisuje na tablicy 1

- Wzajemne wykluczanie,
- Blokada niemożliwa,

Wady:

- Procesy nie są luźno połączone
- Serializacja: P1, P2, P1, P2
- Straty gdy jeden proces chce wchodzić częściej niż drugi
- Proces może zatrzymać się poza sekcją krytyczną i drugi proces też stanie

Współprogramy – raz jeden działa raz drugi (resume(P) – zatrzymaj siebie i wznów proces P)

Rozwiązanie 2:

Jedna tablica: wolne zajęte.

P1, P2:

- Czyta tablicę:
- Zapisuje „zajęte”
- Wchodzi do sekcji krytycznej
- Po wyjściu zapisuje „wolne”

Nie ma zagwarantowanego wykluczania:

- P1 i P2 jednocześnie pod tablicą.
- Czyta tablicę + Zapisuje zajęte - to 2 rozdzielne czynności

Semafory:

- Łatwe w implementacji
- Semafor s zmienna całkowita, przybierająca wartości nieujemne
- Wait (s) – P(s)
- Signal (s) – V(s)
- Wait(s) : jeżeli s>0, to s:=s-1, w przeciwnym przypadku wykonanie procesu, który wywołał Wait(s) zostaje zawieszone
- Signal(s): jeżeli jakiś proces P jest zawieszony na semaforze s na skutek wcześniejszego wykonania Wait(s), to reaktywuj go, w przypadku przeciwnym s:=s+1

- Semafor binarny s={0, 1}

Wzajemne wykluczanie przy użyciu semaforów:

```
Program wzajemne wykluczanie;

Var   s     :     semaphore;   (* binarny*)

Procedure   p1;
      Begin
            Repeat
                  Wait(s);
                        Kryt1;
                  Signal(s);
              Lok1;
            Forever;
      End;

Procedure   p2;
      Begin
            Repeat
                  Wait(s);
                        Kryt2;
                  Signal(s);
              Lok2;
            Forever;
      End;

Begin (*program główny*)
      s:=1;
            Cobegin
                  p1;     p2;
            Coend
End.
```

Wnioski:

- Semafor rozwiązuje problem
- Można dołożyć więcej procesów:
- Ale uwaga: można zagłodzić procesy.