Programowanie współbieżne - wstęp

Problem sortowania *N* liczb

- Algorytm sekwencyjny

```
sortuj(1, N);
```

koszt sortowania przez prostą zamianę - $N^2/2$ porównań

- Algorytm równoległy

```
cobegin    {wykonaj równolegle}
  sortuj(1, pół_N);
  sortuj(pół_N, N);
coend;
 scalaj(1, pół_N, N )
```

koszt algorytmu równoległego - $(N^2/8) + N$ porównań

- Algorytm rekurencyjny, który na każdym poziomie rekurencji wykonuje równolegle dwa sortowania:

- koszt: $1 + 3 + 7 + 15 + ... + (N-1) < 2N$
- wymaga użycia $N/2$ procesorów

- *Złożoność iloczynowa* algorytmu:

rozmiar (liczba procesorów) $\times$ czas

Algorytmy sortowania równoległego:

- zrównoleglone sortowanie przez scalanie: $O(N^2)$
- sieć sortująca parzysto-nieparzyście: $O(N \times (\log N)^4)$
- optymalna sieć sortująca: $O(N \times \log N)$

Sieć o stałych połączeniach: każdy procesor jest połączony ze stałą liczbą procesorów sąsiadujących

W *sieci parzysto-nieparzystej* z każdego komparatora korzysta się tylko raz

Sieci systoliczne: podczas pojedynczego przebiegu procesory są uaktywniane wielokrotnie

Równoległość i potencjalna równoległość

- na poziomie sprzętu
- na poziomie oprogramowania

Programowanie współbieżne

- Notacja i techniki do wyrażania potencjalnej równoległości
- Problem synchronizacji i komunikacji

```
cobegin
  sortuj(1, pół_n);
  sortuj(pół_n, n);
coend;
  scalaj(1, pół_n, n )
```

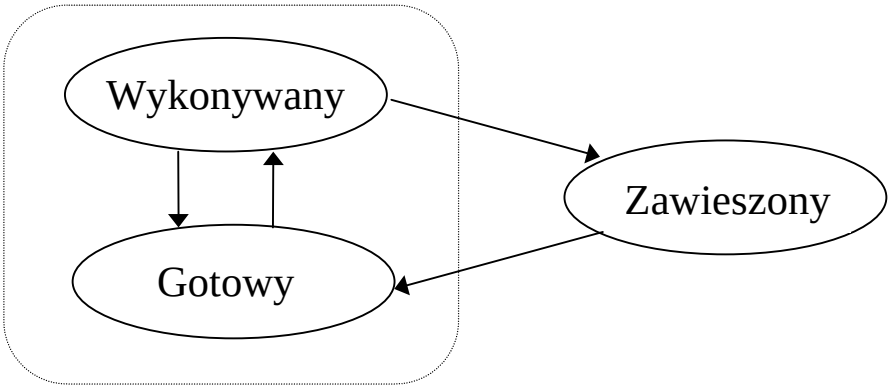
- Równoległość może nie tylko usprawnić działanie programu, ale również poprawić jego jakość (pewne problemy programistyczne są z natury równoległe)

```
cobegin
  while(TRUE) {  /* proces 1 */
    czytaj znak z klawiatury;
    pisz go do portu szeregowego;
  }
  while (TRUE) {  /* proces 2 */
    czytaj znak z portu szeregowego;
    pisz go na ekran;
  }
coend;
```

Procesy współbieżne

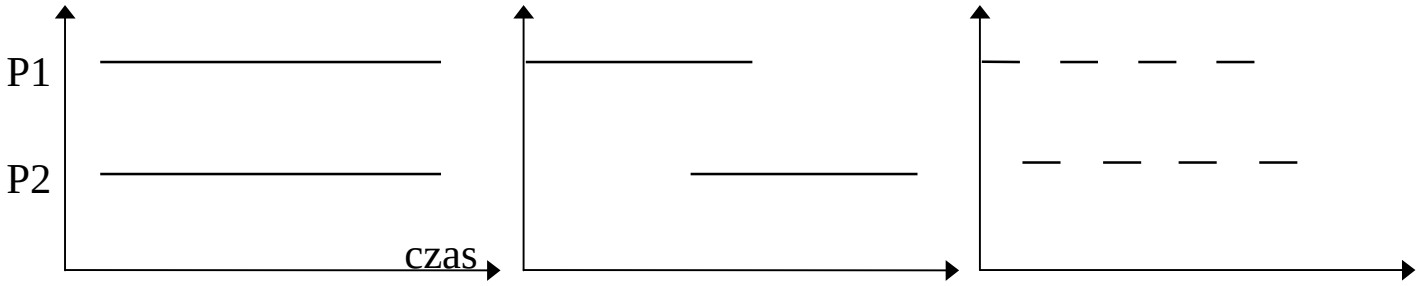
- *Proces sekwencyjny*: realizacja programu w określonym środowisku bądź wstrzymana realizacja w oczekiwaniu na pewne zdarzenie
- Cechy procesu sekwencyjnego:

- obiekt aktywny, który ma przydzielone zasoby
- stany

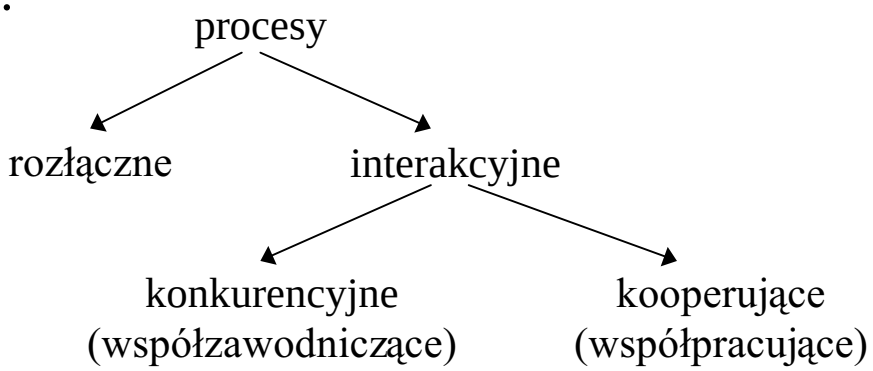


Proces ma przydzielony procesor wirtualny

- Dwa procesy sekwencyjne są *współbieżne*, jeśli wykonywanie jednego z nich zaczęło się po rozpoczęciu, ale przed zakończeniem drugiego



- *Współbieżność* jako *abstrakcja równoległości*
- *Program współbieżny* składa się z kilku (co najmniej dwóch) programów sekwencyjnych, których ciągi wykonawcze są przeplecione. Procesy te zwykle muszą się *komunikować*, aby *synchronizować* działanie lub *wymieniać dane*
- Klasyfikacja:



Poprawność programów współbieżnych

- Nieprzydatność testowania (każdorazowe wykonanie błędnego programu może dać inny wynik)

- Ciąg wykonawczy programu współbieżnego: klasa przeplecionych ciągów wykonawczych procesów składowych
- Dla wykazania niepoprawności wystarczy podać jeden scenariusz przeplatania operacji, dla którego jest źle

Własności programów

- *Własność zapewniania* (bezpieczeństwa, ang. *safety*)

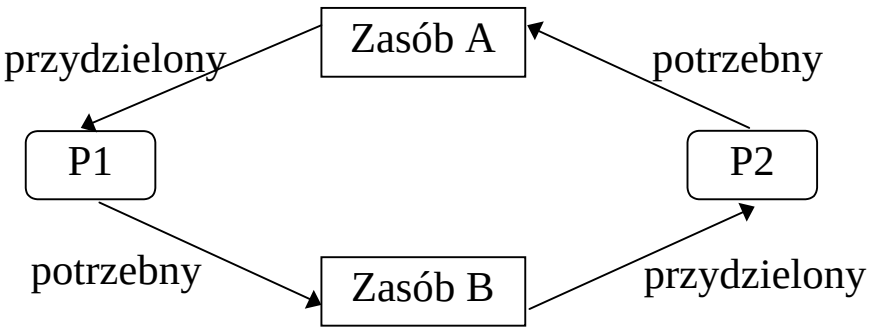
- analogia do częściowej poprawności w programowaniu sekwencyjnym
- wynika z jawnej i statycznej specyfikacji programowanego problemu

przykład: żądanie wzajemnego wykluczania rejonów krytycznych (warunek bezwzględny, nie zmienia się w czasie wykonywania programu)

- *Własność żywotności* (ang. *liveness*)

- analogia do całkowitej poprawności (własność stopu)
- dotyczy dynamicznych aspektów: każde oczekiwane zdarzenie kiedyś nastąpi
- przykłady naruszenia: *zastój*, *zagłodzenie*

Zastój (ang. *deadlock*): każdy proces w pewnym zbiorze procesów czeka na zdarzenie, które może być spowodowane tylko przez proces z tego zbioru



Zagłodzenie (ang. *starvation*): proces jest (nieskończenie długo) wstrzymywany, gdyż zdarzenie, na które czeka, zawsze powoduje wznowienie innego procesu

- mniej groźne - procesor nadal wykonuje sensowną pracę
- zwykle trudniejsze do wykrycia

- *Sprawiedliwość*: nie definiujemy formalnie; czasami będziemy celowo od niej odstępować (priorytety!)
- Pomocny formalizm: *logika temporalna*

  $\Box$ p oznacza, że p jest *zawsze* prawdziwe

  $\Diamond$ p oznacza, że p *w końcu* będzie prawdziwe

Podane operatory ułatwiają wnioskowanie o stwierdzeniach, których prawdziwość zmienia się w czasie

- Udowodnienie poprawności programu współbieżnego wymaga wykazania, że są spełnione zarówno własności zapewniania, jak i żywotności

Schemat procesu

repeat
operacje lokalne
protokół wstępny
rejon krytyczny
protokół końcowy
forever

- *Operacje lokalne* - faza nieistotna z punktu widzenia synchronizacji. Proces może się zatrzymać w swojej sekcji lokalnej, ale nie może to zakłócić pracy pozostałych procesów
- *Rejon krytyczny* - faza wymagająca synchronizacji. Przeważnie chodzi o *wzajemne wykluczanie* się rejonów (własność zapewniania). Jeśli kilka procesów próbuje wejść do swoich rejonów krytycznych, to jednemu musi się to w końcu udać (brak zastoju). Jeśli proces chce wejść do rejonu krytycznego, to w końcu wejdzie (brak zagłodzenia)

Protokoły - konstrukcje synchronizujące; problem języka specyfikacji. Muszą być krótkie i efektywne

Przyjmowane założenia

- Procesy są *słabo powiązane* (ang. *loosely connected*): większość czasu proces poświęca na swoje operacje lokalne, a w rejonie krytycznym przebywa krótko i okazjonalnie
- Nie wolno nic zakładać o bezwzględnych i względnych szybkościach wykonywania
- Jedyne założenia dotyczące czasu, które się przyjmuje, to:

- żaden proces nie pozostaje nieskończenie długo w swoim rejonie krytycznym
- jeśli są procesy gotowe, to w skończonym czasie jeden z nich przejdzie w stan wykonywania

Systemy scentralizowane i rozproszone

- Dzielone zmienne globalne jako reprezentacja pamięci wspólnej w systemach scentralizowanych i wieloprocesorach
- Komunikacja procesów poprzez wysyłanie i odbieranie komunikatów w systemach rozproszonych

Klasyczne problemy współbieżności

- Problem producenta i konsumenta

Przykład z życia: fabryka, magazyn, odbiorca
Przykład z systemów liczących: użytkownik komputera, bufor klawiatury, system operacyjny
Warunki poprawności:
- Konsument pobiera porcję wyprodukowaną przez producenta
- Producent czeka na wolne miejsce w buforze
- Konsument czeka na pełne miejsce w buforze

Warianty:
- bez bufora
- bufor jednoelementowy
- bufor N-elementowy

- bufor nieograniczony (producent nie czeka)
- wielu producentów
- wielu konsumentów
- porcje różnej wielkości

Zajmujemy się synchronizacją producenta i konsumenta. Problem doboru wielkości bufora jest poza naszymi zainteresowaniami

• Problem czytelników i pisarzy

Przykład: rezerwacja miejsc lotniczych

Warunki poprawności:
- Gdy pisarz zapisuje, nikt inny nie czyta, ani nie pisze
- Gdy czytelnik czyta, inni czytelnicy mogą czytać

Warianty:
- priorytet czytelników (dopuszczenie zagłodzenia pisarzy)
- priorytet pisarzy (dopuszczenie zagłodzenia czytelników)
- równe priorytety (bez możliwości zagłodzenia)
- ograniczona liczba miejsc w czytelni

• Problem pięciu filozofów

Przykład abstrakcyjny, ale ukazujący podstawowe problemy współbieżności

Warianty narodowe:
- włoski: filozofowie jedzą makaron widelcami
- chiński: filozofowie jedzą ryż pałeczkami

Warunki poprawności:
- Filozof je, jeśli ma oba widelce
- Żaden widelec nie może być jednocześnie w posiadaniu dwóch filozofów

Warianty:
- widelce podnoszone po kolei (możliwość blokady)
- widelce podnoszone jednocześnie (możliwość zagłodzenia)
- dodatkowo lokaj