ORiSR Pytania Egzaminacyjne

Oznaczenia:

-- - mało wyczerpująca odpowiedź

* - brak odpowiedzi w pdfie z wykładu, definicja z internetu

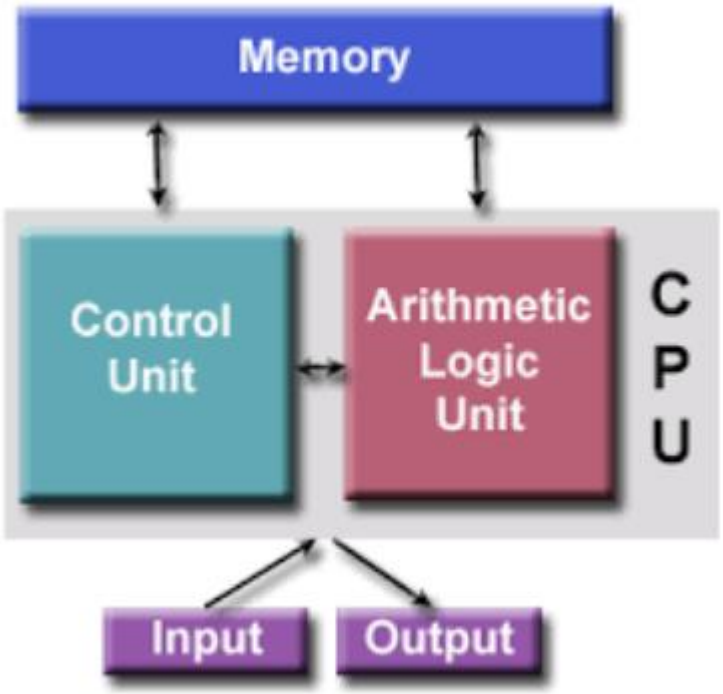
(*) pytanie zadane zaocznym

(*) Wykład 1:

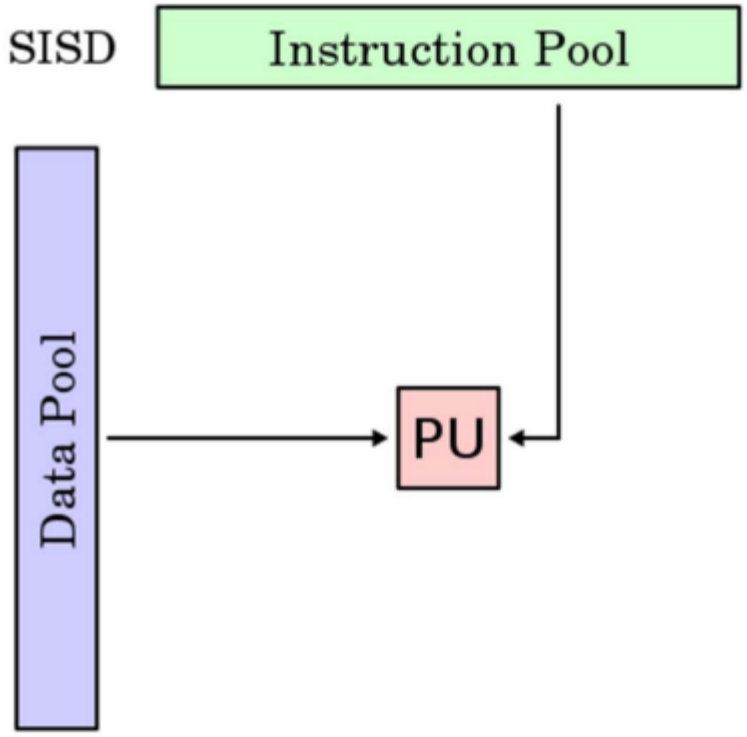
1. Opisać architekturę von Neumana

W architekturze tej komputer składa się z czterech głównych komponentów:

- pamięci komputerowej przechowującej dane programu oraz instrukcje programu; każda komórka pamięci ma unikatowy identyfikator nazywany jej adresem
- jednostki sterującej odpowiedzialnej za pobieranie danych i instrukcji z pamięci oraz ich sekwencyjne przetwarzanie
- jednostki arytmetyczno-logicznej odpowiedzialnej za wykonywanie podstawowych operacji arytmetycznych.
- urządzeń wejścia/wyjścia służących do interakcji z operatorem

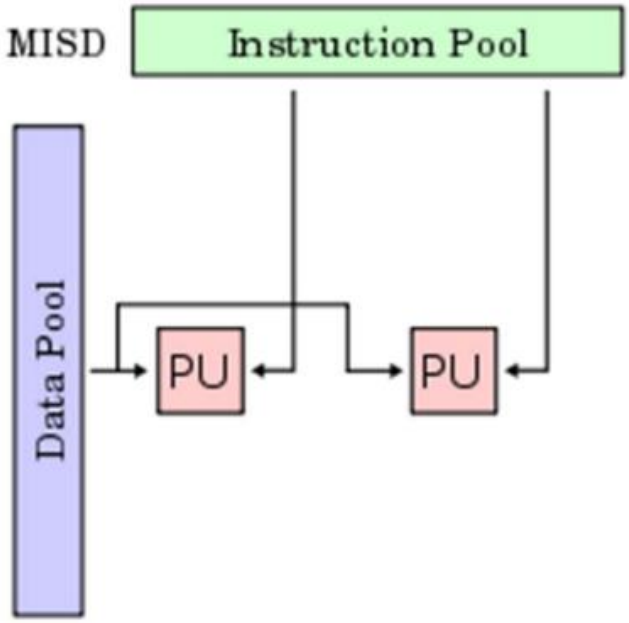


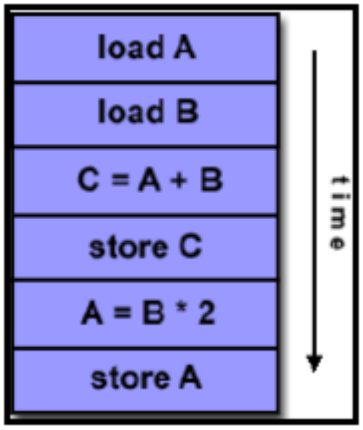
2. Omowić klasyfikacje komputerów wg taksonomii Flynna



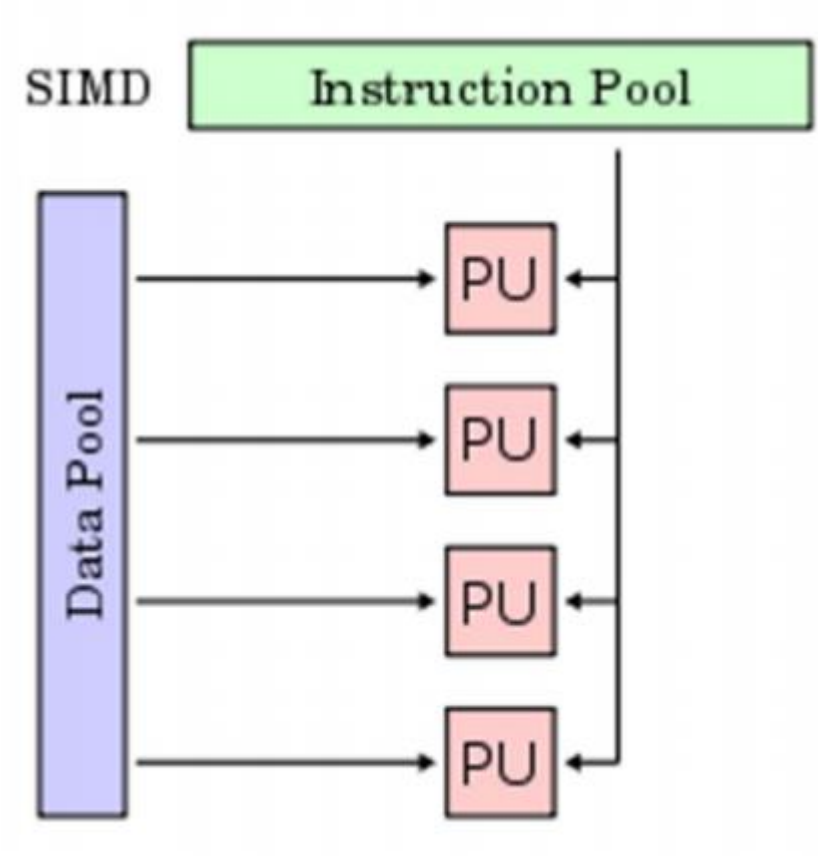
Klasyfikacja architektur komputerowych, zaproponowana w latach sześćdziesiątych XX wieku przez Michaela Flynna, opierająca się na liczbie przetwarzanych strumieni danych i strumieni rozkazów.

SISD (ang. single instruction, single data) - przetwarzany jest jeden strumień danych przez jeden wykonywany program - komputery skalarne (sekwencyjne) - Płyty główne, stacje robocze i komputery starej generacji.

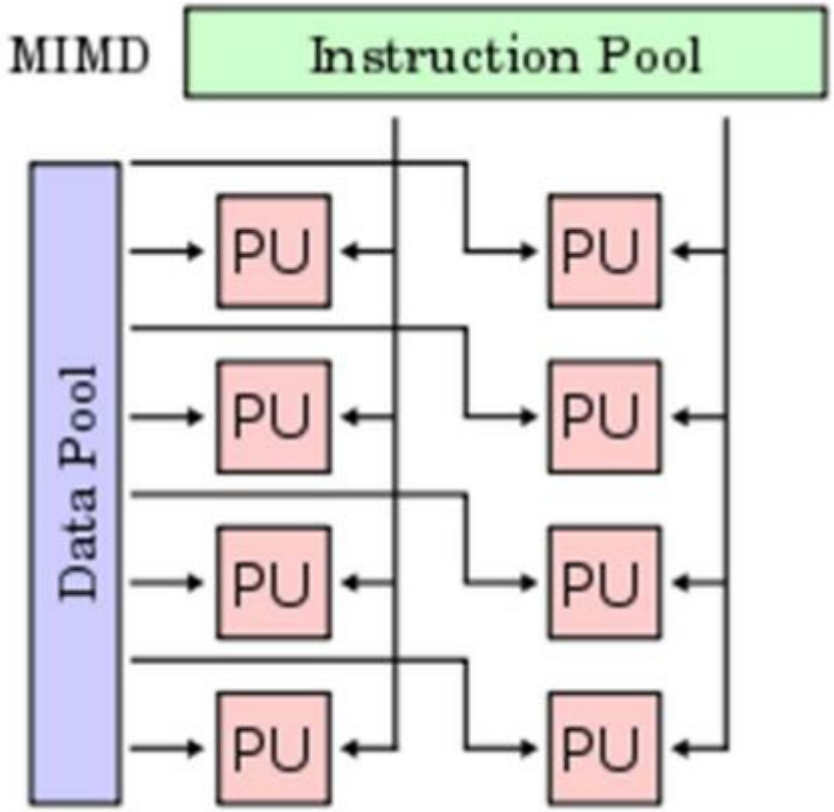




SIMD (ang. single instruction, multiple data) - przetwarzanych jest wiele strumieni danych przez jeden wykonywany program - tzw. Komputery wektorowe.



MISD(ang. multiple instruction, single data) - wiele równolegle wykonywanych programów przetwarza jednocześnie jeden wspólny strumień danych. W zasadzie jedynym zastosowaniem są systemy wykorzystujące redundancję (wielokrotne wykonywanie tych samych obliczeń) do minimalizacji błędów.



MIMD (ang. multiple instruction, multiple data)

- równolegle wykonywanych jest wiele

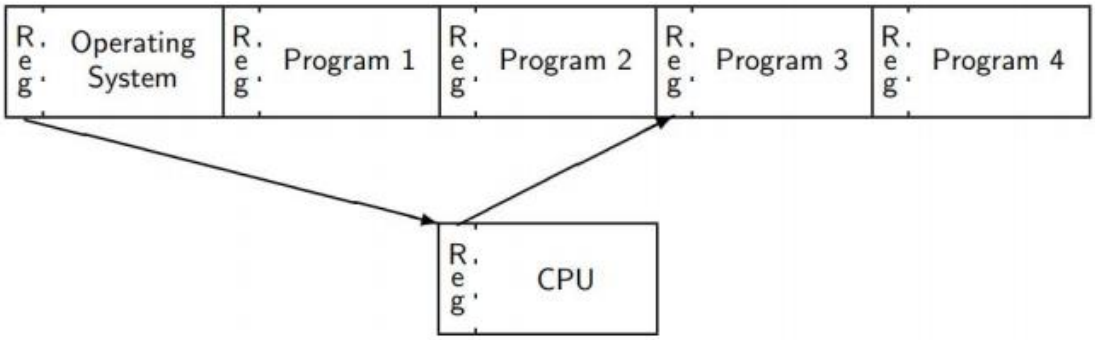
programów, z których każdy przetwarza własne strumienie danych

- przykładem mogą być współczesne komputery wieloprocesorowe i wielordzeniowe, a także klastry i gridy.

3. Opisz znane ci architektury systemowe

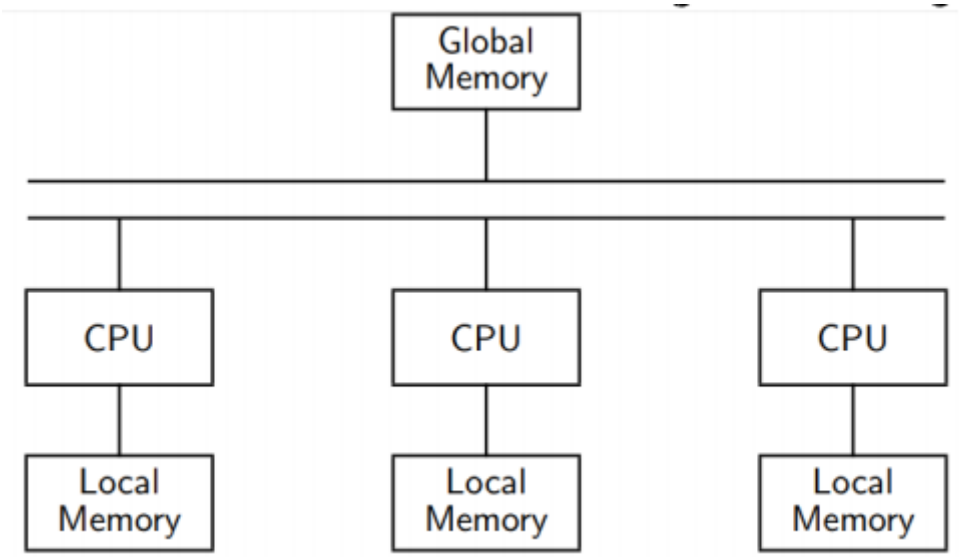
- Systemy wielozadaniowe
- Multiprocesory ze wspólną pamięcią
- Architektura rozproszona

Systemy wielozadaniowe:



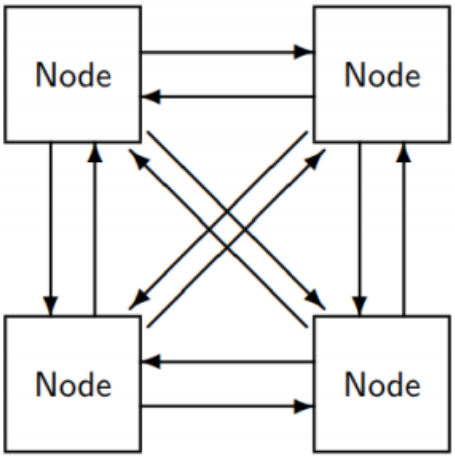
- Nie ma możliwości wykonywania wielu rozkazów równolegle
- O kolejności wykonywanych rozkazów decyduje procesor i system operacyjny
- Występują przerwania przez urządzenia WE/WY
- Po przerwaniu system operacyjny wybiera kolejny proces do wykonania

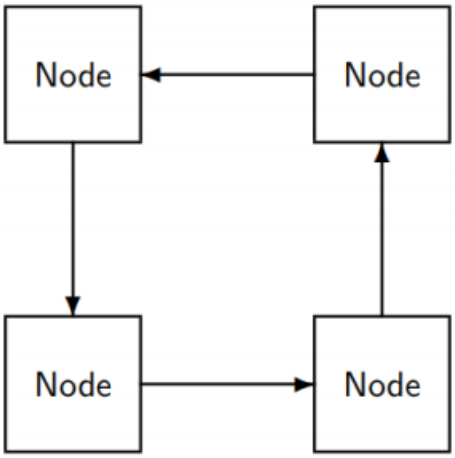
Multiprocesory ze wspólną pamięcią:



- Posiada więcej niż jeden procesor
- Dostęp do pamięci lokalnej mają tylko pojedyncze procesory, do globalnej wszystkie
- Jeśli mamy dostateczną liczbę procesorów to można każdemu z nich przypisać jeden proces (brak przeplotu)
- Jeśli procesy nie sięgają do pamięci globalnej to nie ma rywalizacji i obliczenia są w pełni równoległe

Architektura rozproszona:





- Składa się z wielu komputerów bez wspólnych zasobów
- Są on połączone kanałami komunikacyjnymi
- Często do ich opisu używa się analogii grafowej
- Mogą rozporoszone geograficznie
- Współpraca z innymi węzłami odbywa się poprzez wymianę komunikatów
- Brak pamięci globalnej wspólnej dla wszystkich maszyn
- Topologie pełna – efektywna, ale droga (alternatywa topologia pierścienia)

4. Paradygmaty przetwarzania współbieżnego

Współbieżność iteracyjna: Wiele identycznych procesów, z których każdy zawiera jedną bądź więcej pętli (zmienne dzielone, przesyłanie komunikatów).

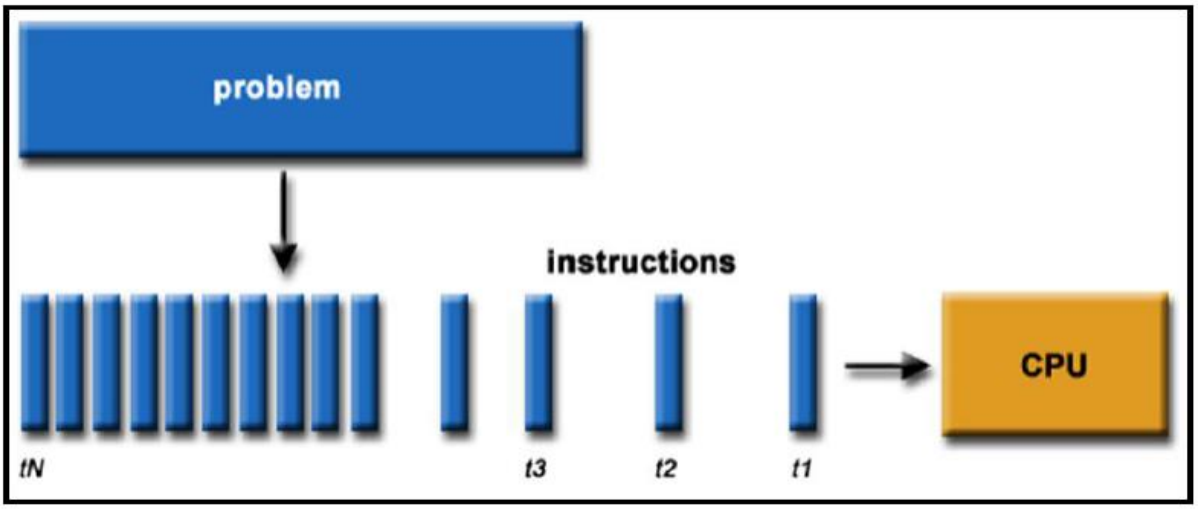
Współbieżność rekurencyjna: Jedna lub więcej procedur rekurencyjnych, wywołania których są niezależne, działających na różnych częściach wspólnych danych.

Producenci i konsumenci (pipelines): Procesy komunikujące się, zorganizowane w potoki, każdy proces jest filtrem wykorzystującym wyjście poprzednika i produkującym wyjście dla następnika.

Klient – serwer: Dominujący wzór interakcji w systemach rozproszonych, proces klient wysyła zamówienie i oczekuje odpowiedzi, proces serwer czeka na zamówienie od klienta i wówczas działa (spotkania, zdalne wywołanie procedur)

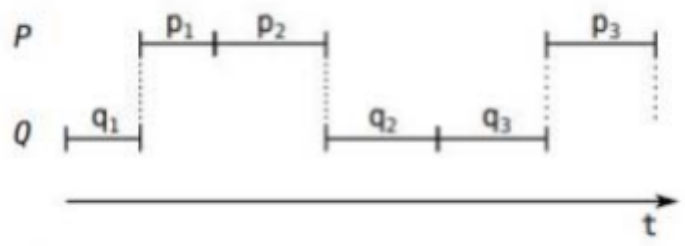
Interaktywne komunikaty: Wiele procesów wykonuje ten sam kod wymieniając się informacjami aby zrealizować zadanie

5. Co to program sekwencyjny, współbieżny, równoległy oraz program działający w przeplocie

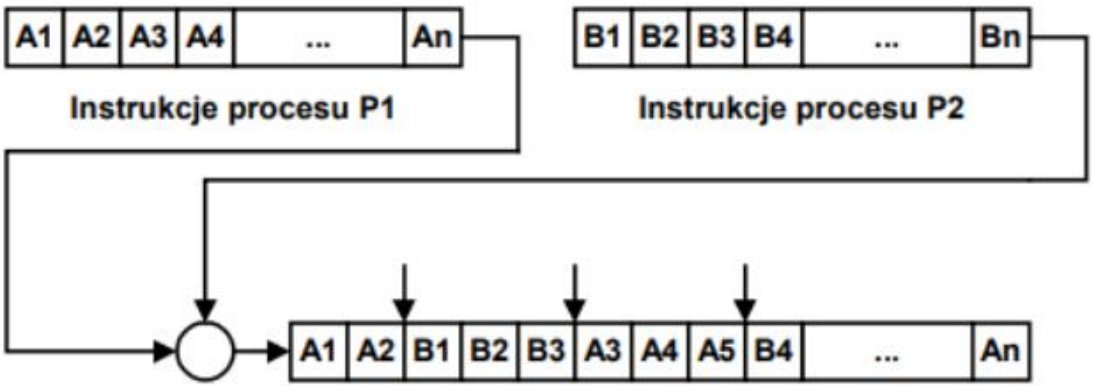


Program sekwencyjny - Poszczególne akcje procesu są wykonywane jedna po drugiej. Dokładniej: kolejna akcja rozpoczyna się po całkowitym zakończeniu poprzedniej.

—

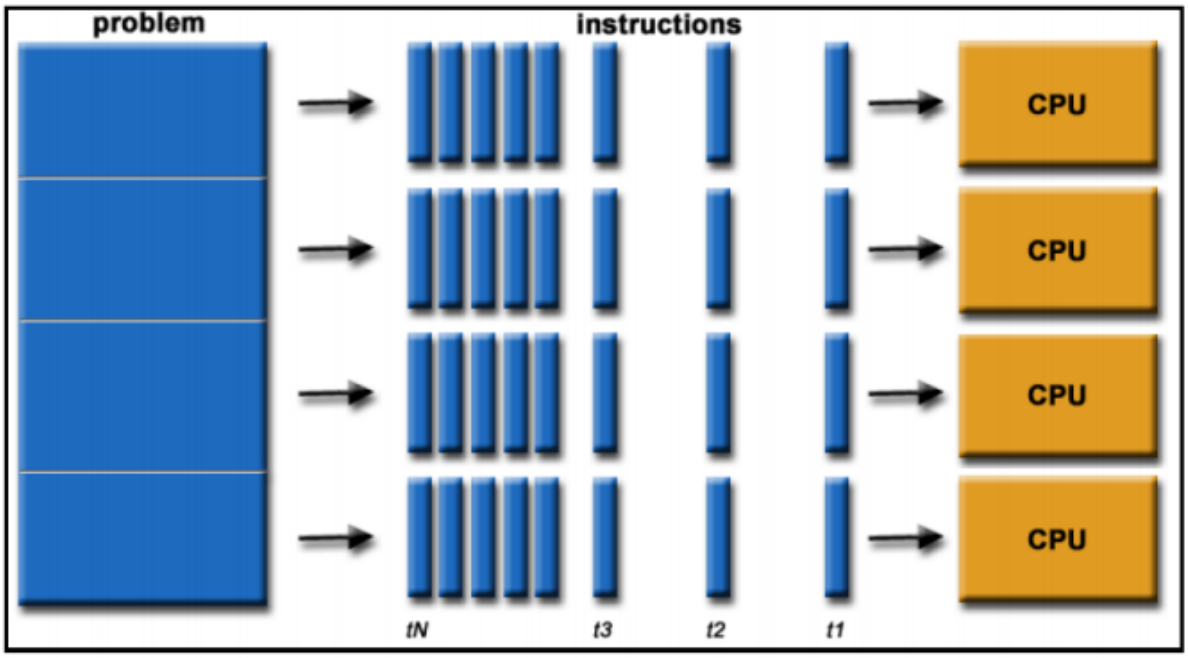


Przelot:



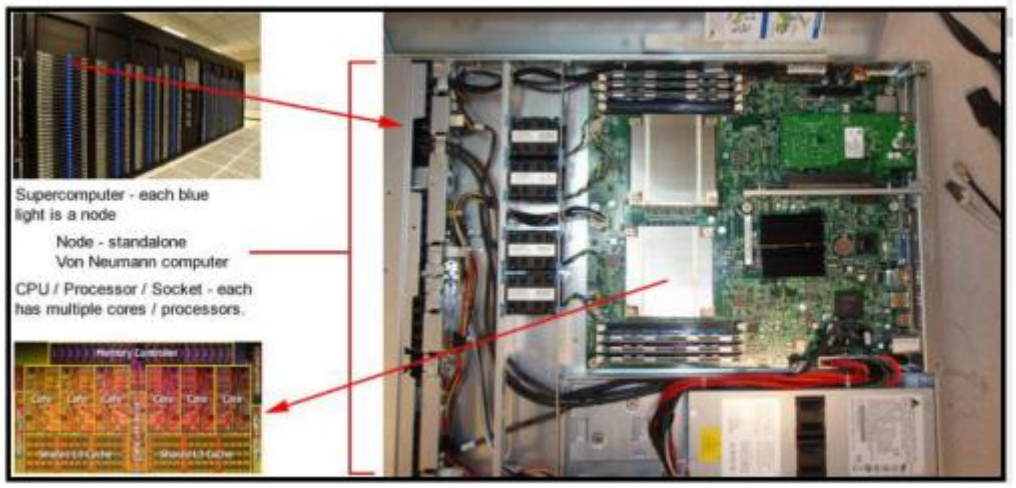
Program działający w przeplocie – Choć jednocześnie odbywa się wykonanie tylko jednej akcji, to jednak jest wiele czynności rozpoczętych i wykonywanych na zmianę krótkimi fragmentami.

—



Program równoległy (ang. parallel program) – odnosi się do systemów, w których wiele programów wykonywanych w tym samym czasie. Obliczenia mogą być realizowane np. na różnych procesorach.

—



Program współbieżny (ang. concurrent program) – zbiór programów sekwencyjnych, które można wykonać równolegle (Dwa procesy są współbieżne jeżeli jeden z nich rozpoczyna się przed zakończeniem drugiego). Jest to uogólnione pojęcie obejmujące zarówno wykonanie równoległe, jak i przeplatane.

6. Przykłady programu sekwencyjnego i równoległego i zapisz go za pomocą pseudo kodu.

Program sekwencyjny

Poniższy fragment kodu przedstawia wycinek algorytmu sortowania przez scalanie tablicy złożonej z N liczb

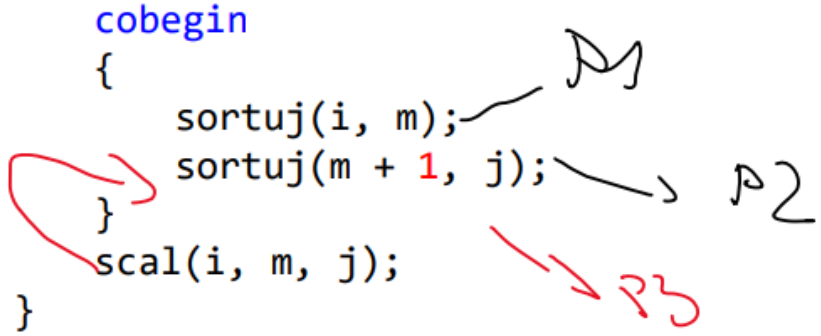
```
void sortuj(int i, int j) // sortuje tablicę A[i..j]
{
    ...
    m = (i + j) / 2;
    sortuj(i, m);
    sortuj(m + 1, j);
    scal(i, m, j);
}
```

Oba wywołania rekurencyjne są niezależne od siebie i dotyczą rozłącznych fragmentów tablicy. Nic nie stoi na przeszkodzie, aby były wykonywane w tym samym czasie: na różnych procesorach lub nawet na jednym z podziałem czasu.

Program równoległy

Dalej wprowadzono słowo kluczowe cobegin oznaczające, że czynności znajdujące się w bloku kodu znajdującym się za nim można wykonywać współbieżnie.

```
void sortuj(int i, int j) // sortuje tablicę A[i..j]
{
    ...
    m = (i + j) / 2;
    cobegin
    {
        sortuj(i, m);
        sortuj(m + 1, j);
    }
    scal(i, m, j);
}
```

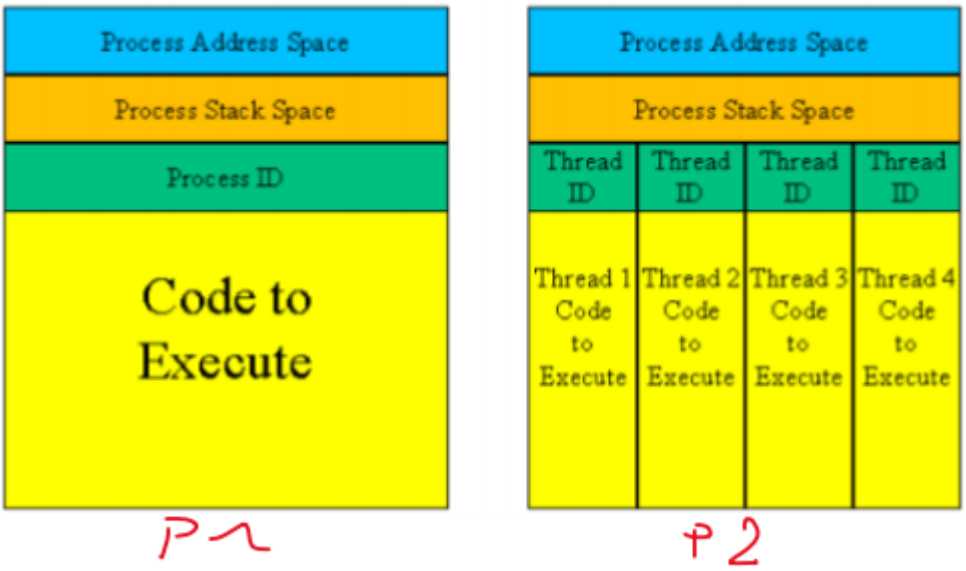


7. Co to jest proces i jakie znasz stany procesu

8. Jaka jest różnica między procesem a wątkiem – wyjaśnić, różnica

Wątek (ang. thread) – część programu wykonywana współbieżnie w obrębie jednego procesu; w jednym procesie może istnieć wiele wątków.

Różnica między zwykłym procesem a wątkiem polega na współdzieleniu przez wszystkie wątki działające w danym procesie przestrzeni adresowej oraz wszystkich innych struktur systemowych (np. listy otwartych plików, gniazd itp.) – z kolei procesy posiadają niezależne zasoby.



9. Na czym polega poprawność częściowa programu sekwencyjnego

W programach sekwencyjnych ponowne uruchomienie programu z tymi samymi danymi wejściowymi zawsze daje ten sam wynik. Usuwanie błędu z takiego programu polega najczęściej na uruchomieniu programu z zastawionymi pułapkami, aż do zdiagnozowania błędu. Następnie kod jest poprawiany i program jest sprawdzany czy wyniki jest poprawny.

W programie współbieżnym niektóre scenariusze prowadzą do poprawnych wyników, inne zaś nie. Nie da się diagnozować programu współbieżnego tak jak to zrobiony powyżej, gdyż każde uruchomienie doprowadzi do uruchomienia innego scenariusz działania aplikacji. Zatem w pewnych sytuacjach błąd może się pojawić, w innych program będzie działać poprawnie.

Poprawność częściowa, definiowana dla programów sekwencyjnych, to własność mówiąca, że jeśli program kończy się, to daje poprawne wyniki.

Programy współbieżne jednak składają się z wielu procesów, z których każdy najczęściej działa w pętli nieskończonej.

Głównym zadaniem programu współbieżnego jest zapewnienie poprawnej synchronizacji przez cały czas działania programu. Tym bardziej nie ma sensu wymaganie, aby program współbieżny kończył się, jak wymaga się tego, przy poprawności całkowitej.

Aby określić poprawność programu współbieżnego definiuje się dwie własności, jakie powinien on posiadać:

- Własność bezpieczeństwa
- Własność żywotności

10. Omówić własność bezpieczeństwa oraz żywotności programu współbieżnego

Własność bezpieczeństwa:

- Nigdy nie dojdzie do niepożądanej sytuacji
- Popatrzmy na przykład. Przypuśćmy, że rozpatrujemy ruchliwe skrzyżowanie w środku dużego miasta. Procesy to samochody usiłujące przez to skrzyżowanie przejechać (na wprost). Własność bezpieczeństwa wyraża fakt, że nie dojdzie do kolizji. Możemy ją wyrazić tak: nigdy na skrzyżowaniu nie będą jednocześnie samochody jadące w kierunku wschód-zachód i północ-południe.
- Własność bezpieczeństwa zazwyczaj jest podawana w specyfikacji zadania, które należy rozwiązać. Stanowi ona odpowiednik częściowej poprawności.
- W omawianym przykładzie skrzyżowania własność bezpieczeństwa można bardzo łatwo zagwarantować: nie wpuszczać na skrzyżowanie żadnego samochodu! Mamy wtedy bardzo bezpieczne rozwiązanie, ale ... trudno uznać je za poprawne. Z tego powodu poprawny program współbieżny powinien mieć także inną własność.

Własność żywotności:

- jeśli proces chce coś zrobić, to w końcu mu się to uda
- W przykładzie skrzyżowania oznacza to, że każdy samochód, który chce przez skrzyżowanie przejechać, w końcu przez to skrzyżowanie przejedzie.

Wykład 2:

1. Co to zasób współdzielony i sekcja krytyczna.

Zasób współdzielony:

Wspólny obiekt, z którego w sposób wyłączny może korzystać wiele procesów.

Sekcja krytyczna:

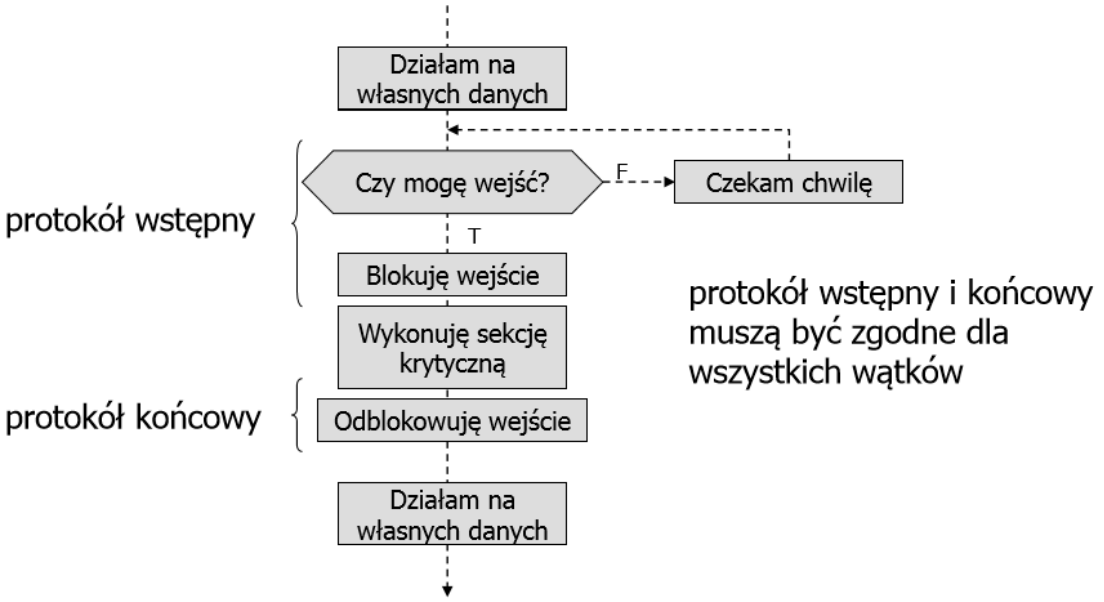
Fragment procesu, w którym korzysta się zasobu dzielonego.

- Między programami – pamięć dyskowa (pliki), drukarka, mysz, klawiatura, usługi systemowe
- Między wątkami – pamięć operacyjna

2. Opisać problem wzajemnego wykluczenia.

- Zsynchronizować N procesów, z których każdy w nieskończonej pętli na przemian zajmuje się własnym sprawami i wykonuje sekcję krytyczną, w taki sposób, aby wykonanie sekcji krytycznych dwóch lub więcej procesów nie pokrywało się w czasie.
- Rozwiązanie powyższego problemu wymaga wprowadzania dodatkowych instrukcji poprzedzających (protokół wstępny) oraz w występujących tuż po zakończeniu sekcji krytycznej (protokół końcowy).
- Protokoły te są implementacją stosowanej w życiu zasady uprzejmości.

PROBLEM WZAJEMNEGO WYKLUCZENIA(2)

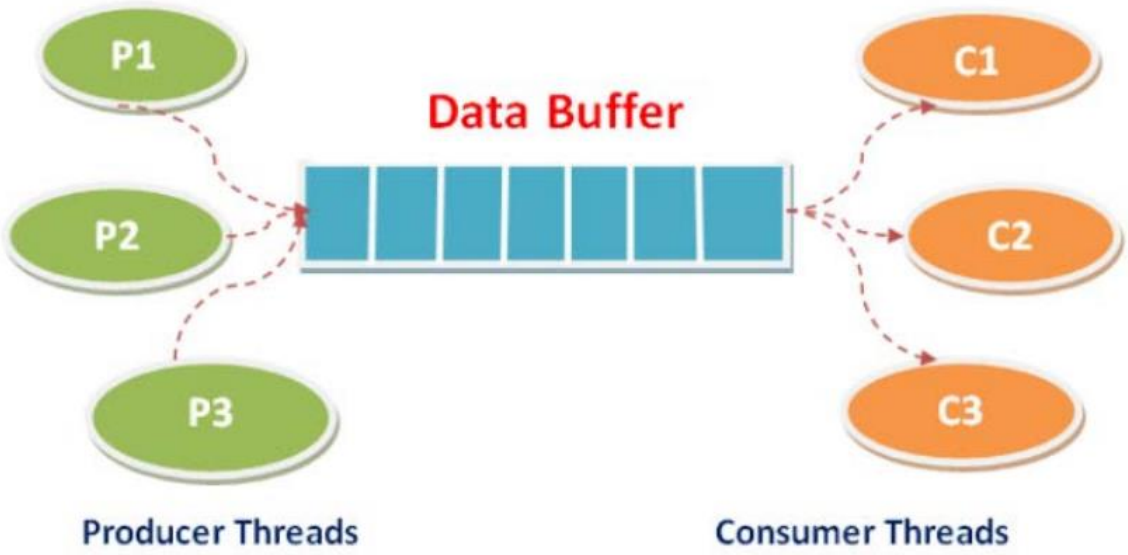


Wymagania czasowe:

- Dotyczą one procesów występujących w procesie wzajemnego wykluczania.
- Żaden proces nie może wykonywać swej sekcji krytycznej nieskończenie długo (nie może się zapętlić lub zakończyć w wyniku jakiegoś błędu). Proces w sekcji krytycznej powinien przebywać jak najkrócej.
- Zachowanie procesów poza sekcją krytyczną nie powinno być w żaden sposób ograniczone.
- Procesy mogę się wykonywać z różnymi prędkościami.

3. Opisać problem producentów i konsumentów.

* Inaczej problem ograniczonego buforowania. *



- W systemie działa P>0 procesów, które produkują pewne dane oraz K>0 procesów, które odbierają dane od producentów.
- Między producentami a konsumentami znajduje się bufor o pojemności B, którego zadaniem jest równoważenie chwilowych różnic w czasie działania procesów.
- Konsument oczekuje na pobranie danych w sytuacji, gdy bufor jest pusty. Gdybyśmy pozwolili konsumentowi odbierać dane z bufora zawsze, to w sytuacji, gdy nikt jeszcze nic w buforze nie zapisał, odebrana wartość byłaby bezsensowna.
- Producent umieszczając dane w buforze nie nadpisywał danych już zapisanych, a jeszcze nie odebranych przez żadnego konsumenta. Wymaga to wstrzymania producenta w sytuacji, gdy w buforze nie ma wolnych miejsc.
- Jeśli wielu konsumentów oczekuje, aż w buforze pojawią się jakieś dane oraz ciągle są produkowane nowe dane, to każdy oczekujący konsument w końcu coś z bufora pobierze. Ni e
  zdarzy się tak, że pewien konsument czeka w nieskończoność na pobranie danych, jeśli tylko ciągle napływają one do bufora.
- Jeśli wielu producentów oczekuje, aż w buforze będzie wolne miejsce, a konsumenci ciągle coś z bufora pobierają, to każdy oczekujący producent będzie mógł coś włożyć do bufora. Nie zdarzy się tak, że pewien producent czeka w nieskończoność, jeśli tylko ciągle z bufora coś jest pobierane.
- Przykład tego problemu jest zapis danych do bufora klawiatury przez sterownik klawiatury i ich odczyt przez system operacyjny.
- Rozpatruje się różne warianty tego problemu:
  - Bufor może być nieskończony.
  - Bufor cykliczny może mieć ograniczoną pojemność.
  - Może w ogóle nie być bufora.
  - Może być wielu producentów lub jeden.

- Może być wielu konsumentów lub jeden.
- Dane mogą być produkowane i konsumowane po kilka jednostek na raz.
- Dane muszą być odczytywane w kolejności ich zapisu lub nie.

4. Opisać problem czytelników i pisarzy.

- W systemie działa C>0 procesów, które odczytują pewne dane oraz P>0 procesów, które zapisują te dane.
- Procesy zapisujące będziemy nazywać pisarzami, a procesy odczytujące - czytelnikami, zaś moment, w którym procesy mają dostęp do danych, będziemy nazywać pobytem w czytelni.
- Jednocześnie wiele procesów może odczytywać dane.
- Jednak jeśli ktoś chce te dane zmodyfikować, to rozsądnie jest zablokować dostęp do tych danych dla wszystkich innych procesów na czas zapisu. Zapobiegnie to odczytaniu niespójnych informacji (na przykład danych częściowo tylko zmodyfikowanych).

```
void Czytelnik() {
        do
        {
                własne_sprawy();
                protokół_wstępny_czytelnika();
                CZYTANIE();
                protokół_końcowy_czytelnika();
        } while (true);
}

void Pisarz() {
        do
        {
                własne_sprawy();
                protokół_wstępny_pisarza();
                PISANIE();
                protokół_końcowy_pisarza();
        } while (true);
}
```

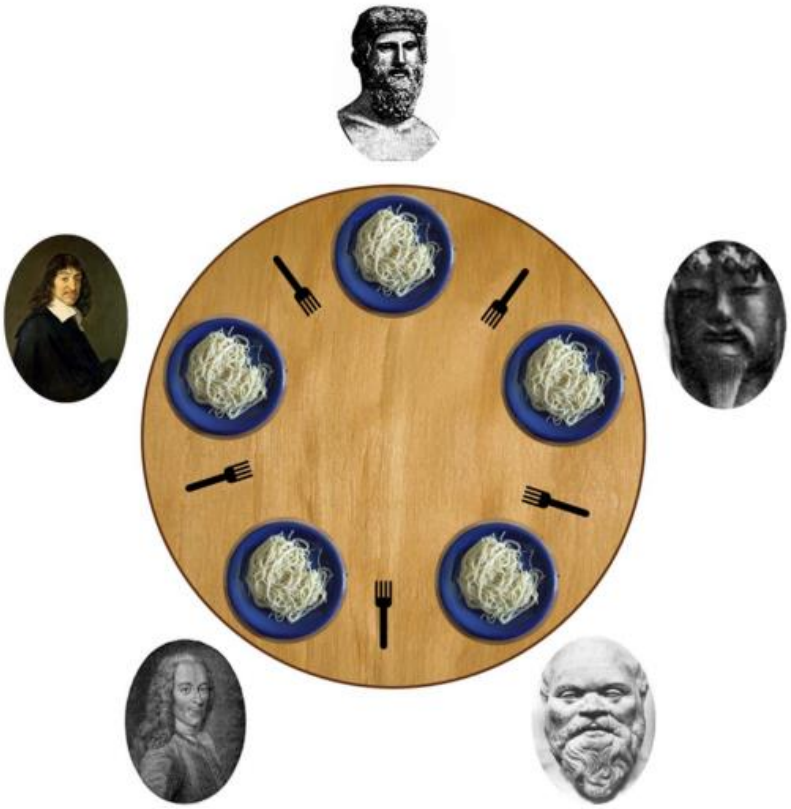
Należy tak napisać protokoły wstępne i końcowe poszczególnych procesów, aby:

- Wielu czytelników powinno mieć jednocześnie dostęp do czytelni.
- Jeśli w czytelni przebywa pisarz, to nikt inny w tym czasie nie pisze ani nie czyta.
- Każdy czytelnik, który chce odczytać dane, w końcu je odczyta.
- Każdy pisarz, który chce zmodyfikować dane, w końcu je zapisze.

Rozpatruje się różne warianty tego problemu:

- W czytelni może przebywać dowolnie wielu czytelników.
- Czytelnia może mieć ograniczoną pojemność.
- Pisarze mogą mieć pierwszeństwo przed czytelnikami (ale wtedy rezygnujemy z żywotności czytelników).
- Czytelnicy mogą mieć pierwszeństwo przed pisarzami (ale wtedy rezygnujemy z żywotności pisarzy).

5. Opisać problem ucztujących filozofów.



- Pięciu filozofów siedzi przy stole i każdy wykonuje jedną z dwóch czynności – albo je, albo rozmyśla.
- Stół jest okrągły, przed każdym z nich znajduje się miska ze spaghetti, a pomiędzy każdą sąsiadującą parą filozofów leży widelec, a więc każda osoba ma przy sobie dwie sztuki – po swojej lewej i prawej stronie.
- Jedzenie potrawy jest trudne przy użyciu jednego widelca, zakłada się, że każdy filozof korzysta z dwóch.
- Nie ma możliwości skorzystania z widelca, który nie znajduje się bezpośrednio przed daną osobą.
- Problem ucztujących filozofów jest czasami przedstawiany przy użyciu ryżu, który musi być jedzony dwiema pałeczkami.
- Gdy filozofowie nigdy nie rozmawiają ze sobą istnieje zagrożenie zakleszczenia w sytuacji, gdy każdy z nich zabierze lewy widelec i będzie czekał na prawy (lub na odwrót).
- Może wystąpić również zagłodzenie, gdy zostanie wzięta pod uwagę kwestia czasu oczekiwania filozofa na dwa wolne widelce.
- Należy tak napisać protokoły wstępne i końcowe, aby:
    - Jednocześnie tym samym widelcem jadł co najwyżej jeden filozof.
    - Każdy filozof jadł zawsze dwoma (i zawsze tymi, które leżą przy jego talerzu) widelcami.
    - Żaden filozof nie umarł z głodu.

6. Scharakteryzować modele współbieżności.

- Model scentralizowany
  - Zmienne globalne i problemy z nimi
  - Niepodzielność instrukcji wysokopoziomowych
  - Mechanizmy synchronizacyjne
- Model rozproszony
  - Komunikacja synchroniczna
  - Komunikacja asynchroniczna
  - Modele przekazywania komunikatów

Model scentralizowany:

- Podstawową cechą modelu scentralizowanego jest możliwość stosowania zmiennych, które są dostępne do zapisu i odczytu przez wiele procesów. Takie zmienne będziemy nazywać zmiennymi dzielonymi lub zmiennymi współdzielonymi.
- Istnienie takich zmiennych w kontekście wykonujących się jednocześnie procesów może powodować pewne problemy. Trzeba na przykład odpowiedzieć na pytanie: co się dzieje, jeśli dwa procesy w tym samym momencie próbują zmodyfikować tę samą komórkę pamięci?
- Z reguły taki konflikt jest rozstrzygany przez układ sprzętowy, noszący nazwę arbitra pamięci, który w jakiś sposób szereguje takie żądania. Mamy więc gwarancję, że na skutek wykonania takiego "jednoczesnego" przypisania, zmienna dzielona będzie miała jedną z wartości, które próbowano ustawić, a nie jakąś przypadkową nieznaną nam wartość.

Zmienne globalne i problemy z nimi:

- Załóżmy dalej, że istnieją dwa procesy P. Każdy z nich pięciokrotnie:
  - Odczytuje zmienną globalną x kopiując jej wartość na swoją zmienną lokalną y.
  - Zwiększa wartość zmiennej lokalnej o jeden.
  - Zapisuje zmodyfikowaną wartość z powrotem na zmienną globalną.

```
double x;
void P() {
        int y, i;
        for (size_t i = 0; i < 5; i++)
        {
                y = x;
                y++;
                x = y;
        }
}
```

- Skutkiem działania przedstawionego kodu jest zwiększenie zmiennej współdzielonej o pięć. I faktycznie, jeśli uruchomimy jeden z tych procesów, to tak się stanie. Podobnie jeśli uruchomimy drugi proces P po zakończeniu działania przez pierwszy proces - wtedy zmienna zwiększy się o 10.
- Co jednak zdarzy się, jeśli procesy uruchomimy współbieżnie? Powiedzmy, że początkową wartością zmiennej x jest 0. Jaka będzie wartość końcowa? Otóż okazuje się, że możemy otrzymać, w zależności od przeplotu akcji poszczególnych procesów, dowolną wartość między 5 a 10.

Podsumowanie:

- Pisząc program współbieżny nie wolno nic założyć o niepodzielności instrukcji języka, w którym ten program tworzymy.
- Model scentralizowany jest trudniejszy dla programisty. Trudniej jest zapewnić poprawność programu współbieżnego korzystającego ze zmiennych współdzielonych. Na dalszej części wykładu omówione będą mechanizmy synchronizacyjne rozwiązujące ten problem. Wśród nich omówione będą semafory, monitory i muteksy.

Model rozproszony:

- Model rozproszony stosuje się najczęściej wtedy, gdy procesy składające się na program współbieżny wykonują się na różnych komputerach. Nie ma wtedy fizycznie miejsca, w którym można by składować zmienne współdzielone. Z tego powodu procesy w tym modelu komunikują się wyłącznie poprzez wymianę komunikatów.
- O modelu rozproszonym można mówić również w sytuacji, gdy procesy działają na tym samym komputerze, ale ich przestrzenie adresowe tworzą logicznie odrębne jednostki i domyślnie procesy nie mogą współdzielić zmiennych globalnych. Takie środowisko wykonania procesów udostępnia m.in. system operacyjny Unix.
- W modelu rozproszonym możemy mówić o komunikacji synchronicznej i asynchronicznej. Różne są też modele przekazywania/adresowania komunikatów oraz identyfikacji procesów.

7. Opisać różnice między komunikacją synchroniczną a asynchroniczną.

Komunikacja synchroniczna:

- Z komunikacją synchroniczną mamy do czynienia wtedy, gdy chcące się ze sobą skomunikować procesy są wstrzymywane do chwili, gdy komunikacja będzie się mogła odbyć. Omówmy ten schemat na przykładzie procesów, z których jeden (zwany nadawcą) chce wysłać komunikat, a drugi (odbiorca) chce komunikat odebrać.
- Jeśli odbiorca chce otrzymać komunikat od nadawcy, to wstrzymuje swoje działanie, aż nadawca będzie chciał go wysłać. Najczęściej oznacza to po prostu oczekiwanie do chwili, aż sterowanie w procesie nadawcy dojdzie do instrukcji wysyłania.
- Podobnie dzieje się, jeśli nadawca chce wysłać komunikat do odbiorcy. Czeka wówczas, aż odbiorca będzie gotowy do odebrania tego komunikatu i dopiero go nadaje.
- Zauważmy, że ponieważ procesy oczekują na siebie, to przesłanie komunikatu może się odbyć bezpośrednio między zainteresowanymi, tzn. nie potrzebny jest żaden bufor, który przechowa wysłany komunikat do momentu jego odbioru przez odbiorcę.

Komunikacja asynchroniczna:

- Komunikacja asynchroniczna nie wymaga współistnienia komunikujących się procesów w tym samym czasie. Polega na tym, że nadawca wysyła komunikat nie czekając na nic. Komunikaty są buforowane w jakimś miejscu (odpowiada za to system operacyjny lub mechanizmy obsługi sieci) i stamtąd pobierane przez odbiorcę.
- Mamy więc następujący schemat:
    - Nadawca wysyła komunikat nie czekając na nic. Komunikat wędruje do bufora.
    - Odbiorca odbiera komunikat z bufora. Jeśli bufor jest pusty, to odbiorca jest wstrzymywany (werblokująca) lub przekazuje w wyniku błąd (wersja nieblokująca).

(*) 9. Wymienić i opisać modele przekazywania komunikatów.

W modelu rozproszonym istnieje też wiele sposobów przekazywania komunikatów.

- Komunikat może być przeznaczony dla konkretnego procesu. Nadawca zna nazwę odbiorcy, a odbiorca zna nazwę nadawcy. Mówimy wtedy o identyfikacji symetrycznej
- Komunikat może być przeznaczony dla konkretnego procesu. Jedna ze stron zna nazwę procesu, druga nie wie kogo obsługuje. Mówimy wtedy o identyfikacji asymetrycznej. Jest to model charakterystyczny dla architektury klient-serwer.
- Komunikat może być wysyłany po kanale komunikacyjnym. Wysyłający zna nazwę kanału, ale nie wie, do kogo trafi komunikat, bo nie musi wiedzieć z kim jest ten kanał połączony.
- Komunikat może być wstawiany do konkretnego bufora, z którego może go odebrać dowolny proces.

Wykład 3:

1. Definicja semafora oraz wymień i opisz typy semaforów.

- Semafor jest to mechanizm synchronizacji procesów, zaproponowany przez Dijkstrę.
- Semafor jest zmienną całkowitą, która z logicznego punktu widzenia (z punktu widzenia aplikacji) przyjmuje wartości nieujemne (20) lub – W przypadku semaforów binarnych – logiczne. Zmienna semaforowa musi mieć nadaną początkową wartość (oczywiście nieujemną).
- Po nadaniu początkowej wartości zmiennej semaforowej można na niej wykonywać tylko dwa rodzaje operacji:
    - P - opuszczanie semafora (hol. proberen),
    - V - podnoszenie semafora (hol. verhogen).
- operacja opuszczania powoduje zmniejszenie wartości zmiennej semaforowej, a operacja podnoszenia jej zwiększenie. Wykonując operację semaforową, proces może zastać zablokowany (przejść w stan oczekiwania). Typowym przypadkiem jest blokowanie w operacji opuszczania semafora. Operacja opuszczania nie zakończy się do czasu, aż wartość zmiennej semaforowej będzie na tyle duża (być może zostanie zwiększona w międzyczasie), że zmniejszenie jej wartości w wyniku tej operacji nie spowoduje przyjęcia wartości ujemnej. W przypadku semaforów dwustronnie ograniczonych blokowanie może wystąpić również w przypadku podnoszenia semafora.

TYPY SEMAFORÓW

- **Semafor binarny** - zmienna semaforowa przyjmuje tylko wartości true (stan podniesienia, otwarcia) lub false (stan opuszczenia, zamknięcia). Przyjmuje się, że wielokrotne podnoszenie takiego semafora nie zmieni jego stanu - skutkiem będzie zawsze stan otwarcia. W niektórych systemach próba podniesienia otwartego semafora może prowadzić do błędu.

- **Semafor ogólny (zliczający)** — zmienna semaforowa przyjmuje wartości całkowite nieujemne, a jej bieżąca wartość jest zmniejszana lub zwiększana o 1 w wyniku wykonania odpowiednio operacji opuszczenia lub podniesienia semafora. Wartością tego semafora jest liczba operacji podniesienia. Liczba operacji opuszczania semafora powinna być równa liczbie operacji podnoszenia.
- **Semafor uogólniony** — semafor zliczający, w przypadku, którego zmienną semaforową można zwiększać lub zmniejszać o dowolną wartość, podaną jako argument operacji. Zmienna semaforowa musi być nieujemna. Jeśli w wyniku wykonanej operacji zmienna semaforowa przyjmie wartość ujemną proces zostanie zablokowany.
- **Semafor dwustronnie ograniczony** – semafor ogólny, w przypadku którego zmienna semaforowa, oprócz dolnego ograniczenia wartością O, ma górne ograniczenie, podane przy definicji semafora.

2. Klasyczna i praktyczna definicja semafora ogólnego.

DEFINICJA KLASYCZNA SEMAFORA OGÓLNEGO

- Definicja ta została podana przez Dijkstrę i zakłada ona, że semafor jest zmienną całkowitą z dwoma operacjami.
- Jako Semphore będzie dalej oznaczony typ odpowiadający semaforowi ogólnemu.
- **Opuszczenie semafora (wait lub P)**
  - czekaj, aż S > 0, S = S-1
- **Podniesienie semafora (signal lub V)**
  - S = S + 1
- Definicja ta nie spełnia warunku niepodzielności.

PRAKTYCZNA DEFINICJA SEMAFORA OGÓLNEGO (BEN-ARIEGO)

- **Podniesienie**
  - if (są procesy wstrzymane podczas opuszczania S)
    wznów jeden z nich;
    else
    S = S +1;
- **Opuszczenie**
  - if (S>0)
    S=S-1;
    else
    wstrzymaj;
- Spełnia warunek niepodzielności

3. Klasyczna i praktyczna definicja semafora binarnego.

DEFINICJA KLASYCZNA SEMAFORA BINARNEGO

- **Opuszczenie semafora (wait lub P od passeren)**
  - czekaj, aż S = true, S = false
- **Podniesienie semafora (signal lub V od vrijmaken)**
  - S = true

DEFINICJA PRAKTYCZNA SEMAFORA BINARNEGO

- **Podniesienie**
  if (są procesy wstrzymane podczas opuszczania S)
    wznów jeden z nich;
  else
    S = true;
- **Opuszczenie**
  if (S == true)
    S = false;
  else
    wstrzymaj;
- Spełnia warunek niepodzielności

4. Wskazać różnice i podobieństwa między semaforem binarnym i ogólnym.

- Semafor binarny nie jest szczególnym przypadkiem semafora ogólnego, gdyż nie pamięta liczby wykonanych operacji podniesienia.
- Semafor binarny może zastąpić ogólny, gdy realizuje wzajemne wykluczanie.
- Jeśli operacja podniesienia może być wykonana wielokrotnie, to semafor binarny nie może być wtedy stosowany.
- Wielokrotne podnoszenie semafora binarnego powinno zakończyć się błędem oraz zakończeniem aplikacji.

5. Podać definicję semafora dwustronnie ograniczonego.

- Niech S jest zmienną całkowitą SE [ON]
- **Podniesienie**
  if (S == N)
    wstrzymaj;
  else if (są procesy wstrzymane podczas opuszczania S)
    wznów jeden z nich;
  else S = 5+1;
- **Opuszczenie**
  if (S == 0)
    wstrzymaj;
  else if (są procesy wstrzymane podczas opuszczania S)
    wznów jeden z nich;
  else S = S-1;

Wykład 3:

1. Co to jest przyspieszenie? Jakie wartości może przyjmować? W jakich warunkach może wystąpić przyspieszenie liniowe bądź superliniowe.

- Zakładamy stały rozmiar danych N
- T(1) czas obliczeń dla najlepszego algorytmu sekwencyjnego
- T(p) czas obliczeń dla algorytmu równoległego przy pomocy p procesorów
- ***Przyspieszenie*** (ang. speedup) S(p)

$$S(p)=\frac{T(1)}{T(p)}$$

- Idealnie S(p)=p (***przyspieszenie liniowe***), w niektórych sytuacjach (rzadko) S(p)>p (przyspieszenie ***superliniowe*** ang. superlinear). W praktycznych algorytmach S(p)<p.

2. Omówić wydajność programu współbieżnego. Określić wydajność dla programu który ma przyspieszenie liniowe bądź superliniowe.

- Wydajność E(p)

$$E(p)=\frac{S(p)}{p}=\frac{T(1)}{p*T(p)}$$

- Wydajność jest stosunkiem osiągniętego przyspieszenia do idealnego przyspieszenia liniowego. Wydajność równa 1 oznacza przyspieszenie liniowe.

Wydajność (ang. Efficiency)

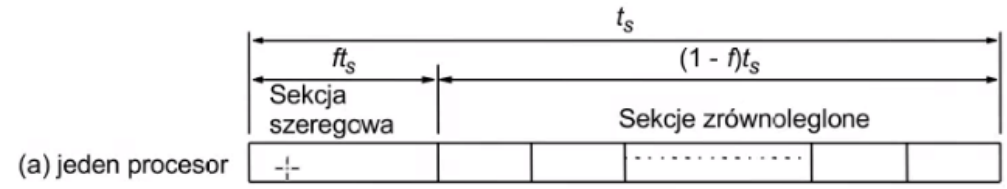
- Wydajność jest miarą użytecznego wykorzystania procesora. Definiuje się ją jako stosunek przyspieszenia do idealnego przyspieszenia liniowego równego p:

$$E(p)=\frac{S(p)}{p}=\frac{T_s}{p*T_p(p)}$$

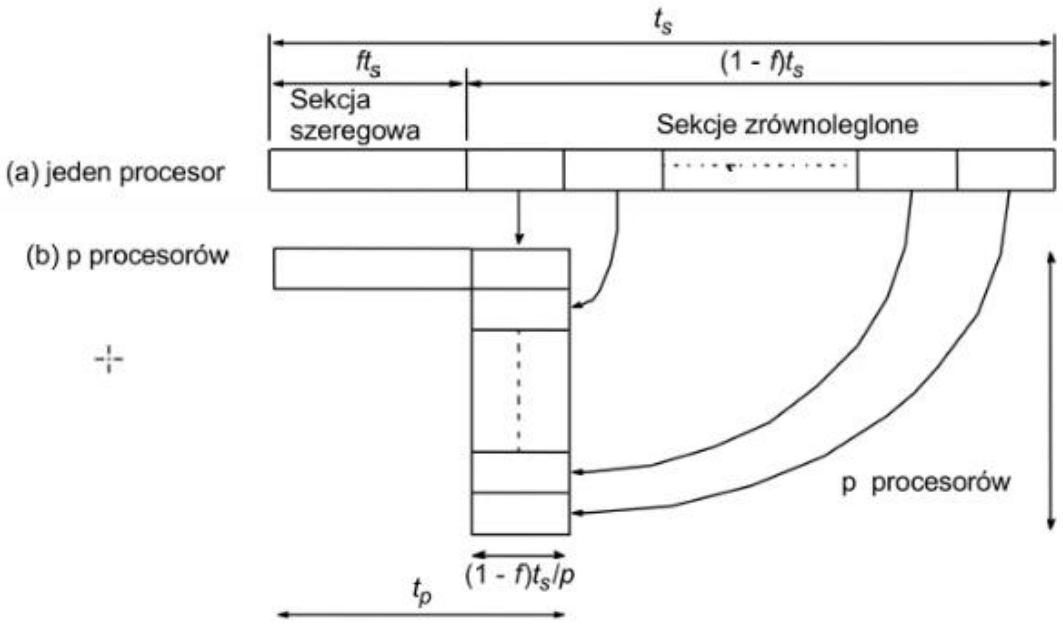
- Wydajność w praktyce zawiera się w przedziale (0,1).
- Wydajność równa 1 oznacza przyspieszenie liniowe.
- W przypadku przyspieszenia superliniowego wydajność jest większa niż 1.

3. Opisać prawo Amdahla, oraz wnioski z niego wynikające.

- G. Amdahl pracownik IBM, założyciel firmy Amdahl.
- Zakładamy że pewna część programu w ogóle nie ulega przyspieszeniu przez równoległe przetwarzanie danych. Pozostała część jest przyspieszona p-krotnie na p procesorach.
- t_s – czas sekwencyjnego wykonania programu. Niech f jest stosunkiem czasu fragmentu nie dającego się przyspieszyć (równego $f*t_s$) do całkowitego czasu t_s.



- t_p – czas obliczeń na p procesorach.



Prawo Amdahla – przyspieszenie

- Przyspieszenie *S(p)* dane jest wzorem:

$$S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1+(p-1)f}$$

zwanym prawem Amdahla.

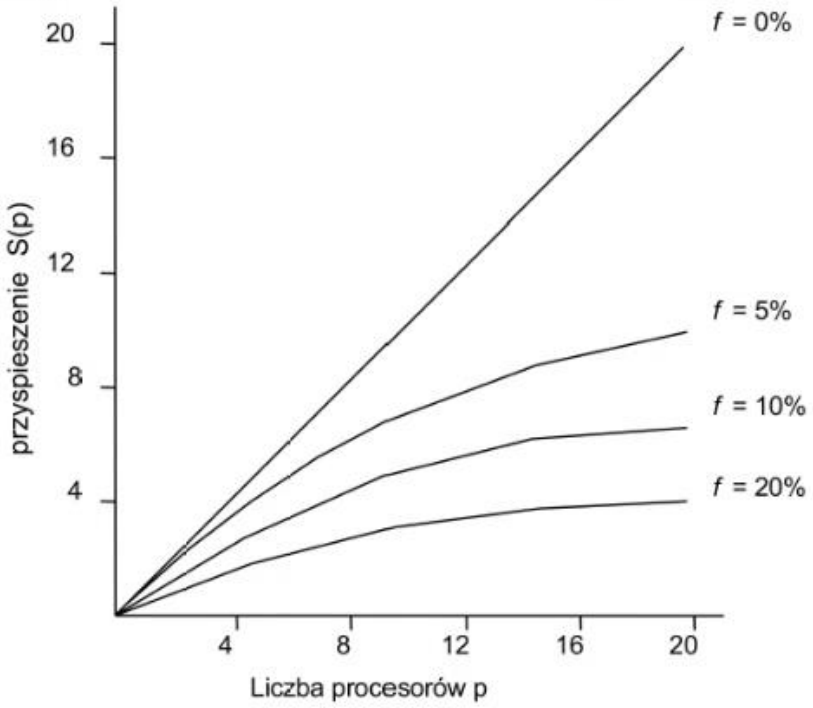
- Uwaga:

$$\lim S(p) = 1/f$$

A zatem dysponując nawet nieskończoną liczbą procesorów, nie przekroczymy przyspieszenia 1/f

Prawo Amdahla – maksymalne przyspieszenie

- dla *f=0 S(p)=p*



Prawo Amdahla - wnioski

- Na pierwszy rzut oka wnioski wydają się pesymistyczne: maksymalne przyspieszenie jest ograniczone z góry.
- W rzeczywistych problemach skalowalność jest ***gorsza*** od wynikającej z prawa Amdahla.
  - Koszty komunikacji procesorów (rosną wraz ze wzrostem ich liczby)
  - Niezrównoważenie obciążenia (ang. load imbalance)
  - Potrzeba duplikacji obliczeń na różnych procesorach.

- Istotnym założeniem w prawie Amdahla jest stały rozmiar problemu.

- To założenie ***nie jest spełnione***. Szybkie komputery buduje się po to aby rozwiązywać coraz bardziej skomplikowane problemy (np. budować złożone modele symulacyjne, grać w bardziej złożone gry)

- Dlatego też dziedzina obliczeń równoległych rozwija się dynamicznie..

4. Omówić źródła narzutów w przetwarzaniu równoległym.

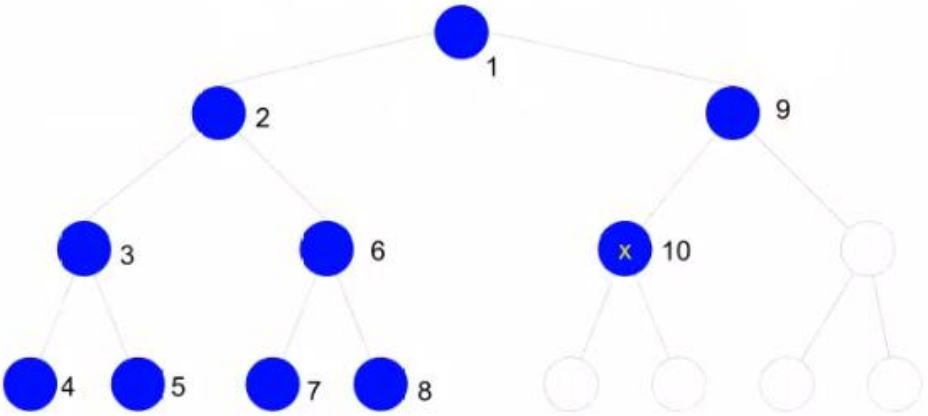
- Czy jeżeli użyję dwóch procesorów to mój program wykona się dwa razy szybciej ?
- ***Z reguły nie***. Program równoległy zawiera wiele dodatkowych narzutów w porównaniu z programem szeregowym. Są to:
- Komunikacja pomiędzy procesorami. Procesory rozwiązujące nietrywialny problem muszą się komunikować ze sobą, co zabiera czas. Koszt ten nie jest ponoszony w wersji szeregowej. Uwaga: koszty komunikacji w modelu ze wspólną pamięcią są ukryte (ale również ponoszone), dlatego tak popularny stał się model programowania z przesyłaniem komunikatów, w którym koszty te są jawne.
- Bezczynność (ang. idling). Procesor może być bezczynny z kilku powodów, np: nierównomiernego podziału pracy pomiędzy procesorami (niezrównoważenie obciążenia - ang. load imbalance), braku zrównoleglenia części algorytmu.
- Nadmiarowe obliczenia. Obliczenia wykonywane przez wersję równoległą, a nie wykonywane przez wersję szeregową. Ich występowanie może być spowodowane trudnościami w zrównolegleniu lub potrzebą minimalizacji komunikacji.

5. Omówić przyczyny przyspieszenia superliniowego (algorytm, graf).

Przyczyna przyspieszenia superliniowego - lepsze wykorzystanie pamięci cache

- Większa łączna pojemność pamięci cache procesorów w systemie wieloprocesorowym prowadzi to większego współczynnika trafień i w konsekwencji co superliniowego przyspieszenia.
- Przykład: procesor z 64KB pamięci cache ma współczynnik trafień 80% (dla danego algorytmu i rozmiaru danych).
- W systemie dwuprocesorowym łączna pojemność pamięci cache dwóch procesorów wynosi 128KB. Ponieważ rozmiar danych pozostaje taki sam współczynnik trafień może wzrosnąć. Przyjmijmy że wzrośnie do 90%. Z pozostałych 10%, 8% to dostępy do pamięci lokalnej DRAM 2% do pamięci zdalnej (zakładamy architekturę NUMA).
- Przyjmując czas dostępu do pamięci cache 2ns, lokalnej 100ns, zdalnej 400 ns dostajemy (obliczenia - A. Grama) przyspieszenie 2.43 dla dwóch procesorów.
- Powyższa analiza zakłada brak jakichkolwiek narzutów, co nie zdarza się w praktyce. W rezultacie przyspieszenie superliniowe, z powodu lepszego wykorzystania pamięci cache zdarza się bardzo rzadko (ale czasami ma miejsce).

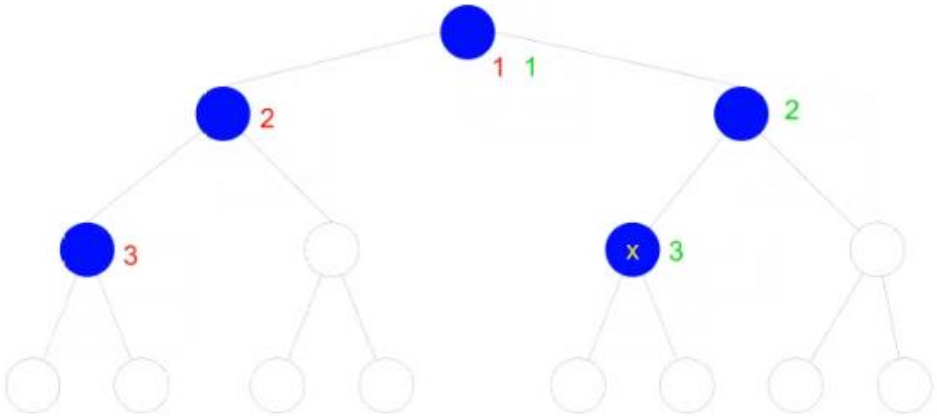
Przyczyna przyspieszenia superliniowego - program szeregowy wykonuje więcej pracy



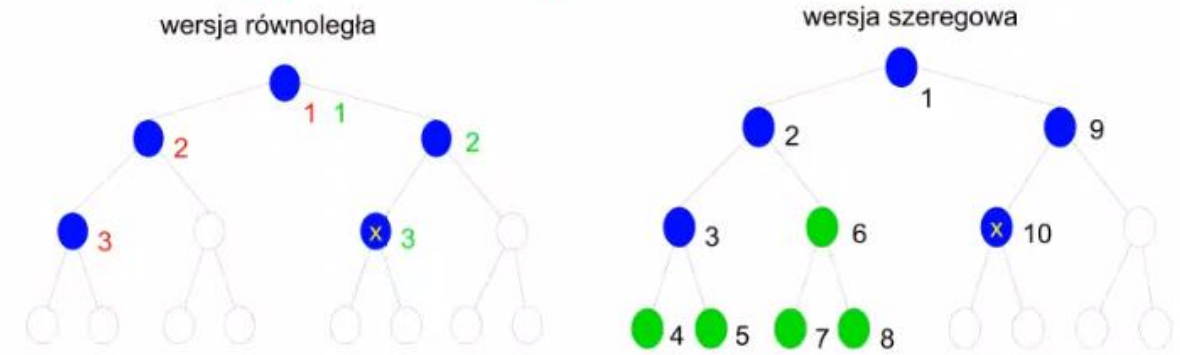
- Przeszukiwanie wgłąb drzewa od korzenia, w poszukiwaniu wierzchołka ze znakiem „x".
- Wierzchołki odwiedzane są zaznaczone na niebiesko, liczby pokazują kolejność odwiedzania
- Razem 10 odwiedzanych wierzchołków.

Przeszukiwanie wgłąb - wersja równoległa

- Wersja dla dwóch procesorów. Obydwa zaczynają od korzenia.
- **Procesor pierwszy** otrzymuje lewe poddrzewo a **procesor drugi** prawe. Obydwa poddrzewa przeszukiwane są równolegle
- Zakładamy, że pierwszy procesor, który znajdzie wierzchołek kończy pracę całego programu.
- W trzecim kroku procesor drugi znajduje „x", przyspieszenie 10/3=3.33 !



Przyspieszenie superliniowe - przyczyna



- Wyraźnie widać, że program równoległy wykonuje mniej pracy.
- Dzięki "lepszemu" punktu startowemu jednego z procesorów natrafiamy szybciej na rozwiązanie i nie ma potrzeby odwiedzania wierzchołków drzewa zaznaczonych na zielono.
- W tej sytuacji algorytm składający się z dwóch procesów na jednym procesorze również szybciej odnalazłby rozwiązanie od algorytmu szeregowego.
  - Kłóci się to z założeniem porównywania z najlepszym algorytmem szeregowym, ale dla tego problemów trudno sformułować algorytm "najlepszy"

12

6. Opisać czym jest koszt algorytmu równoległego. Określić koszt algorytmu równoległego który jest optymalny.

Koszt

- Koszt jest iloczynem czasu pracy i liczby procesor

$$C(p) = p * T_p(p)$$

- Koszt odzwierciedla sumę czasu spędzonego przez wszystkie procesory pracujące nad rozwiązaniem problemu.
- System równoległy jest ***optymalny ze względu na koszt***, jeżeli jego koszt jest tego samego rzędu (jako funkcja rozmiaru danych), co czas pracy najlepszego algorytmu szeregowego.
- Znaczenie praktyczne tego pojęcia: Można wykazać że algorytm optymalny ze względu na koszt ma ***stałą wydajność***.
  - Jeżeli nie jest optymalny ze względu na koszt, wydajność spada przy liczbie procesorów dążącej do nieskończoności.

7. Zdefiniować całkowity narzut programu równoległego. Określić wielkość narzutu w przypadku przyspieszenia liniowego.

Całkowity narzut równoległy (ang. total parallel overhead)

- T_p - czas wersji równoległej, T_s - czas wersji szeregowej, p liczba procesorów.
- Całkowity czas spędzony przez wszystkie procesory pracujące nad rozwiązaniem problemu jest równy pT_p.
- T_s jednostek tego czasu jest spędzona na użytecznej pracy pozostałość jest całkowitym narzutem równoległym (T_o).

$$T_o(p) = p * T_p(p) - T_s$$

8. Rodzaje dekompozycji oraz stopień współbieżności.

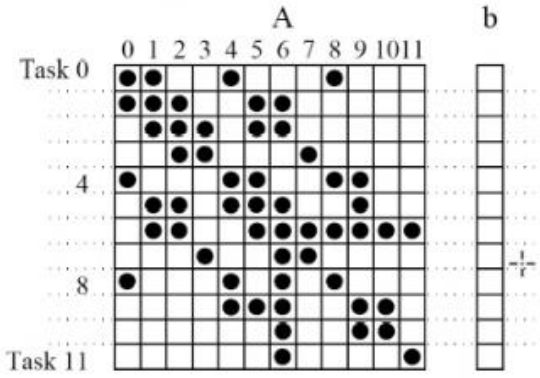
Ziarnistość (ang. granularity) dekompozycji i stopień współbieżności

- Liczba zadań na które dzielimy problem określają nam ziarnistość dekompozycji.
  - Dekompozycja gruboziarnista (ang. coarse grained) - problem podzielony na niewielką liczbę stosunkowo dużych zadań.
  - Dekompozycja drobnoziarnista (ang. fine grained) - problem podzielony na dużą liczbę niewielkich zadań.
- Stopień współbieżności (ang. degree of concurency) - jest liczbą zadań, które mogą być wykonywane równolegle.
  - Może się on zmieniać w trakcie wykonywania programu, stąd wyróżniamy maksymalny stopień współbieżności oraz średni stopień współbieżności.
- Stopień współbieżności zwiększa się, gdy dekompozycja staje się bardziej drobnoziarnista i na odwrót.
  - Ale nie oznacza to, że dekompozycja drobnoziarnista prowadzi do lepszej wydajności. Zwiększanie ziarnistości prowadzi bowiem do większej interakcji pomiędzy zadaniami, co zwiększa czas wykonania programu (koszty komunikacji).

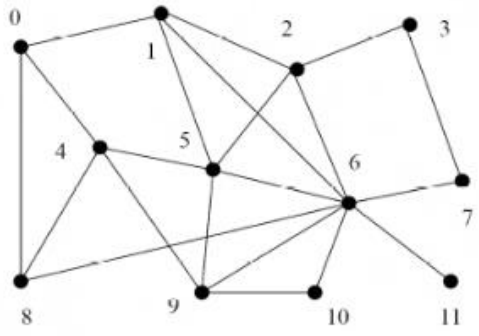
9. Co to jest graf interakcji, do czego można go wykorzystać.

Graf interakcji zadań

- Jest to graf niezorientowany, w którym ścieżka pomiędzy dwoma wierzchołkami reprezentującymi dwa zadania oznacza istnienie interakcji. Graf zależności zadań jest z reguły podgrafem grafu interakcji.
- Przykład: mnożenie macierzy A rzadkiej i wektora b z podziałem wektora b pomiędzy zadania. Każde zadanie odpowiada za jeden element b i wysyła go pozostałym.



macierz rzadka
elementy ze znakiem • są różne od zera



graf interakcji zadań