Rozproszone systemy operacyjne

Anna Kobusińska

Anna.Kobusinska@cs.put.poznan.pl
www.cs.put.poznan.pl/akobusinska

Literatura

1. **J. Bacon**, *Concurrent Systems-An Integrated Approach to Operating Systems*, Addison Wesley, 1992.
2. **J. Brzeziński,** *Ocena stanu globalnego w systemach rozproszonych,* OWN 2001
3. **R. Guerraoui, L. Rodrigues**, *Reliable Distributed Programming*, Springer-Verlag, Berlin, 2006
4. **K. M. Chandy, J. Misra**, *Parallel Program Design, A Foundation,* Addison Wesley, 1988.
5. **A. Gościński**, *Distributed Operating Systems, The Logical Design,* Addison Wesley, 1991.
6. **J.M. Helary, M. Raynal**, *Synchronization and Control of Distributed Systems and Programs,* John Wiley & Sons, 1990.
7. **K. Hwang, F. Briggs**, *Computer Architecture and Parallel Processing,* McGraw Hill, 1984.
8. **J.Jaja**, *An Introduction to Parallel Algorithms,* Addison Wesley, 1992.
9. **A.D. Kshemkalyani, M. Singhal**, *Distributed Computing – Primciples, Algorithms, and Systems,* Cambridge University Press, Cambridge, 2008
10. **N. Lynch**, *Distributed Algorithms,* Morgan Kaufmann Publishers, Inc., 1996.
11. **S.J. Müllender**, *Distributed Systems,* ACM Press, 1993.
12. **M. Raynal**, *Ditributed Algorithms and Protocols,* John Wiley & Sons, 1988.
13. **M. Singhal, N.G. Shivaratri**, *Advanced Concepts in Operating Systems – Distributed, Database, and Multiprocessor Operating Systems,* McGraw Hill, 1994.
14. **A. Tanenbaum, M. van Steen**, *Distributed Systems. Principles and Paradigms*,Prentice Hall, 2002.
15. **G.Tel**, *Introduction to Distributed Algorithms,* Cambridge University Press, 1994.
16. http://distributedcomputing.info/ - portal na temat przetwarzania rozproszonego
17. http://library.thinkquest.org/C007645/ - strona edukacyjna na temat systemów rozproszonych
18. http://www.hyper.net/dc-howto.html - krótka charakterystyka największych projektów DC

Rozwój technologiczny

Postęp w ostatnim półwieczu

- 10 mln dolarów, 1 instrukcja na sekundę
- 1000 dolarów, 100 mln instrukcji na sekundę
- 12^{12} razy lepszy współczynnik cena/efektywność
- sieci komputerowe LAN i WAN

Definicja systemu rozproszonego

System rozproszony — zestaw niezależnych komputerów, sprawiający na jego użytkownikach wrażenie jednego, logicznie zwartego systemu.

Aspekty

- sprzęt: maszyny są autonomiczne
- oprogramowanie: wrażenie pojedynczego systemu

Cechy systemów rozproszonych

- Ukrycie przed użytkownikami:
  - różnic pomiędzy poszczególnymi komputerami
  - sposobów komunikowania się komputerów
  - wewnętrznej organizacji systemu rozproszonego
- Jednolity i spójny interfejs dla użytkownika — niezależnie od czasu i miejsca interakcji
- Łatwość rozszerzania (skalowania)

Architektura systemu rozproszonego

- Rozbudowa własności lokalnych systemów operacyjnych (SO)
- Usługi dla aplikacji rozproszonych
- Oprogramowanie warstwy pośredniej (ang. *middleware*)

Cele systemów rozproszonych

- Łatwe połączenie użytkownik-zasoby
- Ukrywanie faktu rozproszenia zasobów
- Otwartość
- Skalowalność

Łączenie użytkowników i zasobów

- Ekonomia: współdzielony dostęp do zasobów jest tańszy
  - drukarki
  - specjalizowane komputery
  - szybkie pamięci
  - bazy danych
  - zdalne dokumenty
- Bezpieczeństwo

Przezroczystość (I)

System przezroczysty — (transparentny, ang. *transparent*) sprawia wrażenie systemu scentralizowanego (dla użytkowników i aplikacji).

Przezroczystość dostępu (ang. *access transparency*) ujednolicanie metod dostępu do danych i ukrywanie różnic w reprezentacji danych.

Przezroczystość (II)

Przezroczystość położenia (ang. *location transparency*) – użytkownicy nie mogą określić fizycznego położenia zasobu (np. na podstawie jego nazwy czy identyfikatora).

Przezroczystość wędrówki (ang. *migration transparency*) – można przenosić zasoby pomiędzy serwerami bez zmiany sposoby odwoływania się do nich.

Przezroczystość przemieszczania (ang. *relocation transparency*) – zasoby mogą być przenoszone nawet podczas ich używania.

Przezroczystość (III)

Przezroczystość zwielokrotniania (ang. *replication transparency*) ukrywanie przed użytkownikami faktu zwielokrotniania (replikacji) zasobów.

Przezroczystość współbieżności (ang. *concurrency transparency*) możliwość współbieżnego przetwarzania danych nie powodująca powstawania niespójności w systemie.

Przezroczystość (IV)

Przezroczystość awarii (ang. *failure transparent*) maskowanie przejściowych awarii poszczególnych komponentów systemu rozproszonego.

Przezroczystość trwałości (ang. *persistence transparency*) maskowanie sposobu przechowywania zasobu (pamięć ulotna, dysk).

Kompromis pomiędzy dużym stopniem przezroczystości a efektywnością systemu.

Otwartość systemów rozproszonych

- Usługi zgodne ze standardowymi regułami opisującymi ich składnię i semantykę
  - przykład: protokoły komunikacyjne w sieciach komputerowych
- Specyfikacja usług poprzez interfejsy
  - język opisu interfejsu (ang. *Interface Definition Language*)
- Specyfikacje muszą być *zupełne* (kompletne) i *neutralne*
  - zdolność do współdziałania (ang. *interoperability*)
  - przenośność (ang. *portability*)

Systemy elastyczne

Systemy elastyczne

- łatwość konfiguracji
- łatwość rekonfiguracji (np. wymiana poszczególnych komponentów)

Organizacja systemu rozproszonego

- projekt monolityczny
- logiczne wydzielenie składowych
- podział na autonomiczne komponenty (koszt: wydajność)
- oddzielenie polityki od mechanizmu

Skalowalność

Wymiary skalowalności

- skalowalność pod względem rozmiaru
  - decentralizacja: danych, usług i algorytmów
- skalowalność geograficzna
- skalowalność pod względem administracyjnym

Algorytmy zdecentralizowane

- brak informacji globalnej
- decyzje na podstawie informacji lokalnych
- odporność na awarie pojedynczych maszyn
- brak założeń dot. istnienia globalnego zegara

Problemy związane z konstrukcją systemów rozproszonych

- optymalne zrównoleglenie algorytmów przetwarzania
- ocena poprawności i efektywności algorytmów rozproszonych
- alokacja zasobów rozproszonych
- synchronizacja procesów
- ocena globalnego stanu przetwarzania
- realizacja zaawansowanych modeli przetwarzania
- niezawodność
- bezpieczeństwo

Motywy

- różnorodność otwartych problemów związanych z konstrukcją i zarządzaniem systemami rozproszonymi
- ogromne rzeczywiste zapotrzebowanie na systemy rozproszone
- dostępność środków technicznych i praktyczne możliwości realizacji systemów rozproszonych

Klasy zastosowań

- Aplikacje wykorzystujące przetwarzanie rozproszone na wielu jednostkach obliczeniowych (ang. *distributed supercomputing*)
- Aplikacje wymagające dużej przepustowości (ang. *high throughput*)
- Aplikacje „na żądanie” (ang. *on demand*)
- Aplikacje intensywnie przetwarzające dane (ang. *data intensive*)
- Aplikacje umożliwiające współpracę (ang. *collaborative*)

GRID

„A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities ”

/ Ian Foster /

GRID – cechy

dependable consistent **pervasive** **inexpensive**

usługa wiarygodna
użytkownicy żądają pewności, że otrzymają przewidywalny, nieprzerwany poziom wydajności dzięki różnym elementom tworzącym GRID.

usługa powszechnie dostępna (wszechobecna)
usługa zawsze powinna być dostępna, niezależnie od tego gdzie znajduje się użytkownik tej usługi.

usługa spójna
potrzebny jest standardowy serwis, dostępny poprzez standardowe interfejsy, pracujący ze standardowymi parametrami.

usługa relatywnie tania (opłacalna)
dostęp do usługi powinien być relatywnie tani, tak by korzystanie z takiej usługi było atrakcyjne także z ekonomicznego punktu widzenia.

Co GRID powinien …

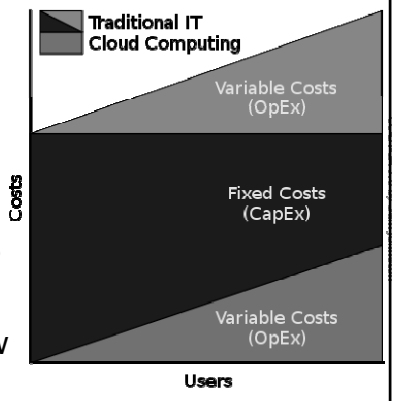
- umożliwiać rozproszenie geograficzne zasobów
- obsługiwać heterogeniczność sprzętową i programową
- być połączony poprzez heterogeniczną sieć
- korzystać z ogólnie dostępnych, standardowych protokołów i interfejsów
- być odporny na zawodny sprzęt
- pozwalać na zmianę dynamiki dostępu do sprzętu
- zrzeszać różne organizacje (wirtualne) z ich własnymi politykami bezpieczeństwa i dostępu do zasobów

Cloud computing

Chmura obliczeniowa (ang. *cloud*) to zbiór zasobów wraz z oprogramowaniem, zarządzanych i udostępnianych z użyciem usług przez dostawcę.

Typowa charakterystyka:

- **skalowalne**, dostępne **na żądanie**
- napędzane **ekonomią**, nie standardami; **pay-as-you-go**
- **łatwe** w użyciu
- dostępne dla wielu użytkowników z pomocą **wirtualizacji**



Cloud Computing Economics

Cloud computing – możliwości i ryzyko

- *„płacisz za tyle ile używasz”*
- eliminacja kosztów inwestycyjnych – infrastruktury… ryzyka
- **automatyzacja** zarządzania przez dostawcę, automatyzacja użycia przez użytkownika (skalowalność na żądanie itp.)
- **iluzja dostępu do nieskończonej puli zasobów** dostępnych na żądanie; dowolne zastosowania

- gdzie są moje dane?
- nad czym mam kontrolę?
- co jeśli jeden dostawca nie działa? ...

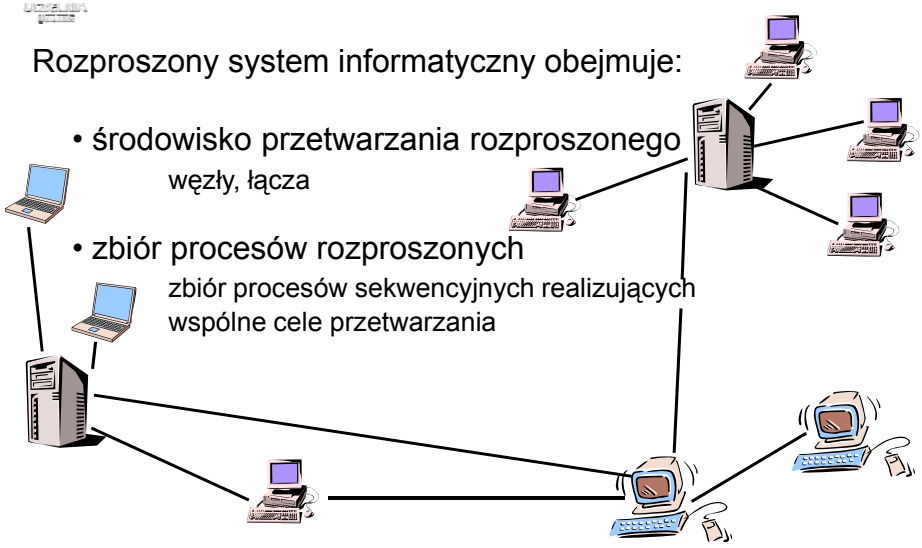
Przykładowe projekty przetwarzania rozproszonego

- SETI
- Cure Cancer
- Fight Anthrax
- Prime Numbers
- Distributed.net
- GIMPS
- FreeDB.org
- The Internet Movie Database
- The Distributed Chess Project
- Wikipedia
- Dmoz – Open Directory Project
- ClimatePrediction.net
- Lifemapper

Rozproszony system informatyczny

Rozproszony system informatyczny obejmuje:

- środowisko przetwarzania rozproszonego
  węzły, łącza
- zbiór procesów rozproszonych
  zbiór procesów sekwencyjnych realizujących wspólne cele przetwarzania



Środowisko przetwarzania rozproszonego

Środowisko przetwarzania rozproszonego jest zbiorem $\mathcal{N}$ autonomicznych jednostek przetwarzających N_i (węzłów), zintegrowanych siecią komunikacyjną (środowiskiem komunikacyjnym, łączami komunikacyjnymi, łączami transmisyjnymi).

Komunikacja w środowisku przetwarzania rozproszonego

Komunikacja między węzłami możliwa jest tylko przez transmisję **pakietów informacji** (wiadomości, komunikatów) łączami komunikacyjnymi.

Zegary w środowisku przetwarzania rozproszonego

Jednostki przetwarzające realizują przetwarzanie z prędkością narzucaną przez **lokalne zegary**.

- zegary są **niezależne** → węzły działają **asynchronicznie**.
- zegary są **zsynchronizowane** lub istnieje **wspólny zegar globalny** dla wszystkich węzłów → węzły działają **synchronicznie**.

Węzeł

Jednostka przetwarzająca $N_i \in \mathcal{N}$ (węzeł) jest elementem środowiska przetwarzania rozproszonego obejmującym:

- procesor
- lokalną pamięć operacyjną
- interfejs komunikacyjny

Łącze komunikacyjne

Łącze komunikacyjne jest elementem umożliwiającym transmisję informacji między interfejsami odległych węzłów. Wyróżnia się łącza jedno- i dwu-kierunkowe.

Bufory łącza

Łącza są wyposażone w bufory o określonej pojemności (ang. *links capacity*).

Jeżeli łącze nie posiada buforów (jego pojemność jest równa zero), to mówimy o łączu **nie buforowanym**, w przeciwnym razie – o **buforowanym**.

Kolejność odbierania komunikatów

- **Łącze FIFO** – kolejność odbierania komunikatów wysyłanych z danego węzła jest zgodna z kolejnością ich wysyłania
- **Łącze nonFIFO** – w przeciwnym przypadku

Niezawodność łącza

Łącza mogą gwarantować również, w sposób niewidoczny dla użytkownika, że żadna wiadomość nie jest tracona, duplikowana lub zmieniana – są to tzw. **łącza niezawodne** (ang. *reliable*, *lossless*, *duplicate free*, *error free*, *uncorrupted*, *no spurious*).

Czas transmisji

Czas transmisji w łączu niezawodnym (ang. *transmission delay*, *in-transit time*) może być ograniczony lub jedynie określony jako **skończony** lecz **nieprzewidywalny**.

Struktura środowiska przetwarzania

Często przedstawiana jako graf:

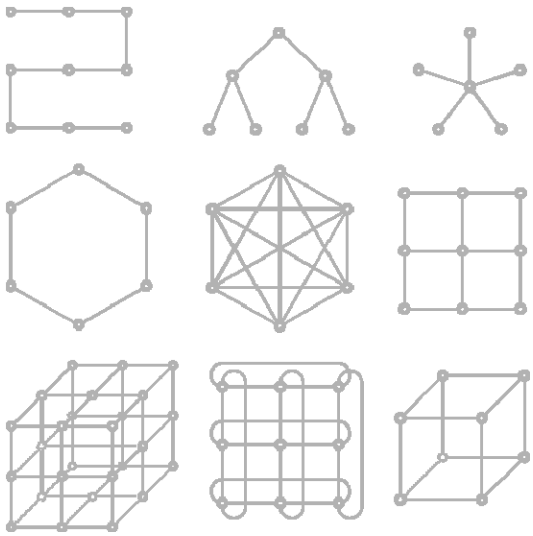
$$\mathcal{G} = \langle \mathcal{V}, \mathcal{A} \rangle$$

w którym :

- wierzchołki grafu $V_i \in \mathcal{V}$ reprezentują jednostki przetwarzające $N_i \in \mathcal{N}$,
- krawędzie $(V_i, V_j) \in \mathcal{A}$, $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$, grafu niezorientowanego lub łuki $\langle V_i, V_j \rangle \in \mathcal{A}$ grafu zorientowanego, reprezentują odpowiednio łącza dwu- lub jedno-kierunkowe.

(35)

Przykłady topologii



(36)

Przetwarzanie rozproszone

Procesem rozproszonym (przetwarzaniem rozproszonym) nazywamy **współbieżne** i **skoordynowane** (ang. *concurrent and coordinated*) wykonanie w środowisku rozproszonym zbioru $\mathcal{P}$ procesów sekwencyjnych P_1, P_2, P_3, ..., P_n współdziałających w realizacji wspólnego celu przetwarzania.

Proces sekwencyjny

Nieformalnie, każdy **proces sekwencyjny** jest działaniem wynikającym z wykonywania w pewnym środowisku (kontekście) programu sekwencyjnego (algorytmu sekwencyjnego), który składa się z ciągu operacji (instrukcji, wyrażeń) atomowych (nieprzerywalnych).

Klasy operacji

Wyróżnia się dwie podstawowe klasy operacji:

- **wewnętrzne** (ang. internal)
  odnoszą się tylko do zmiennych lokalnych programu
- **komunikacyjne** (ang. communication)
  odnoszą się do środowiska i dotyczą komunikatów (ang. *messages*) oraz kanałów (ang. *channels*)

Kanał - definicja

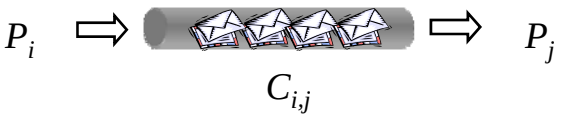
Kanał jest obiektem (zmienną) skojarzonym z uporządkowaną parą procesów $\langle P_i, P_j \rangle$, modelującym jednokierunkowe łącze transmisyjne.

Typem tego obiektu jest zbiór wiadomości, którego rozmiar nazywany jest **pojemnością kanału**.

Kanały incydentne, wejściowe i wyjściowe

Kanał skojarzony z parą procesów $\langle P_i, P_j \rangle$, oznaczamy przez $C_{i,j}$ oraz nazywamy **kanałem incydentnym** z procesem P_i i z procesem P_j.

Ponadto, kanał $C_{i,j}$ nazywamy **kanałem wyjściowym** procesu P_i oraz **kanałem wejściowym** procesu P_j.



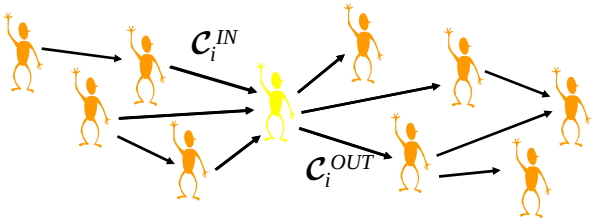
(41)

Zbiory kanałów

Zbiór kanałów wejściowych i wyjściowych tego procesu odpowiednio przez $\mathcal{C}_i^{IN}$ i $\mathcal{C}_i^{OUT}$.
Zbiór wszystkich kanałów incydentnych procesu P_i oznaczymy przez $\mathcal{C}_i$.

Tak więc:

$$\mathcal{C}_i = \mathcal{C}_i^{IN} \cup \mathcal{C}_i^{OUT} \qquad (2.1)$$



(42)

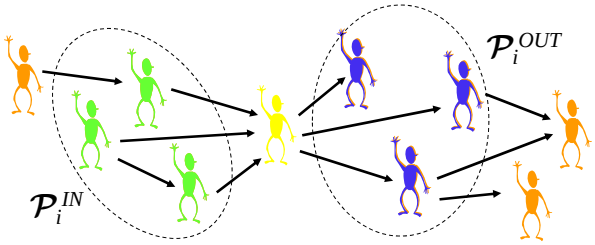
Zbiory procesów sąsiednich

Zbiór sąsiednich procesów wejściowych procesu P_i

$$\mathcal{P}_i^{IN} = \{P_j : \langle P_j, P_i \rangle \in \mathcal{C}_i^{IN}\} \qquad (2.2)$$

Zbiór sąsiednich procesów wyjściowych procesu P_i

$$\mathcal{P}_i^{OUT} = \{P_j : \langle P_i, P_j \rangle \in \mathcal{C}_i^{OUT}\} \qquad (2.3)$$



Stan kanału

Przez ***stan*** $L_{i,j}$ ***kanału*** $C_{i,j}$ rozumieć będziemy zbiór, lub uporządkowany zbiór, wiadomości wysłanych przez proces P_i lecz jeszcze nie odebranych przez proces P_j.

Modelowanie opóźnienia w kanale

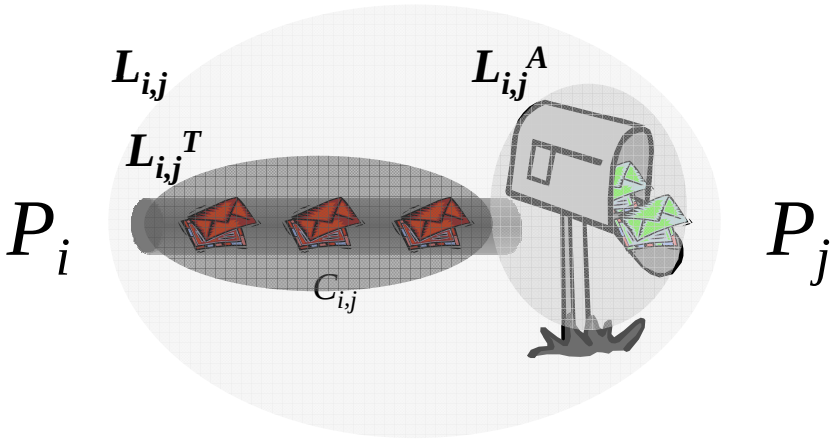
W celu modelowania w kanale opóźnień komunikacyjnych, w zbiorze wiadomości $L_{i,j}$ wyróżnia się dwa rozłączne podzbiory:

- **zbiór wiadomości transmitowanych** $L_{i,j}^{T}$ (ang. *in-transit*)
- **zbiór wiadomości dostępnych** $L_{i,j}^{A}$ (ang. *available*, *arrived*, *ready*)

Oczywiście, w każdej chwili

$$L_{i,j} = L_{i,j}^{T} \cup L_{i,j}^{A} \qquad (2.4)$$

Stan kanału – przykład

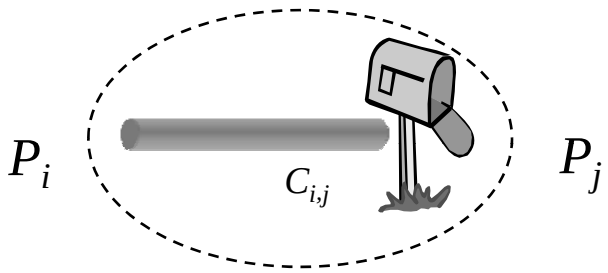


Predykaty opisujące stan kanału

- ***empty*($C_{i,j}$)**
- ***in-transit*($C_{i,j}$)**
- ***available*($C_{i,j}$)**

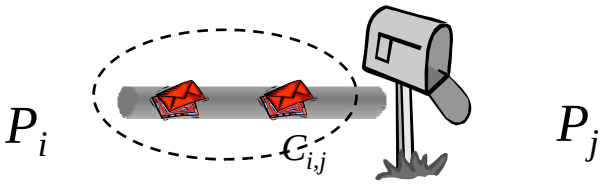
Predykat *empty*

$$\textbf{\textit{empty}}(C_{i,j}) \equiv L_{i,j} = \varnothing \qquad (2.5)$$



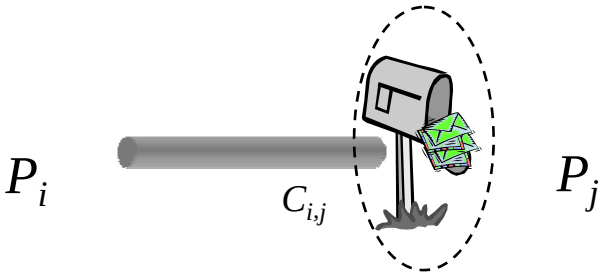
Predykat *in-transit*

$$\textbf{\textit{in-transit}}(C_{i,j}) \equiv \boldsymbol{L_{i,j}}^{T} \neq \varnothing \qquad (2.6)$$



Predykat *available*

$$\textbf{\textit{available}}(C_{i,j}) \equiv \boldsymbol{L_{i,j}}^{A} \neq \varnothing \qquad (2.7)$$



Indywidualne operacje komunikacyjne (1)

***send*(P_i, P_j, M)**

Efektem wykonania tej operacji jest umieszczenie wiadomości M w kanale $C_{i,j}$, a więc wykonanie podstawienia

$$L_{i,j} := L_{i,j} \cup \{M\} \quad (2.8)$$

Indywidualne operacje komunikacyjne (2)

***receive*(P_i, P_j, *inM*)**

Jeżeli kanał $C_{i,j}$ nie jest pusty i pewna wiadomość M jest bezpośrednio dostępna ($available(C_{i,j})$ ma wartość True), to efektem wykonania tej operacji jest pobranie wiadomości M z kanału $C_{i,j}$, a więc wykonanie podstawienia:

$$L_{i,j} := L_{i,j} \setminus \{M\} \text{ oraz } inM := M \quad (2.9)$$

Grupowe operacje komunikacyjne (1)

***send*(P_i, $\mathcal{P}_i^R$, M)**

Efektem wykonania tej operacji jest umieszczenie wiadomości we wszystkich kanałach $C_{i,j}$, takich że $P_j \in \mathcal{P}_i^R$, a więc podstawienie dla wszystkich tych kanałów:

$$L_{i,j} := L_{i,j} \cup \{M\}$$

(2.10)

Grupowe operacje komunikacyjne (2)

***receive*($\mathcal{P}_j^S$, P_j, *sInM*)**

Efektem wykonania operacji ***receive***($\mathcal{P}_j^S$, P_j, $sInM$) jest atomowe pobranie wiadomości M_i od procesów $P_i \in \mathcal{P}_j^S$ i umieszczanie ich w $sInM$. Tym samym, dla każdego procesu $P_i \in \mathcal{P}_j^S$, wykonywane jest kolejno podstawienie

$$L_{i,j} := L_{i,j} \setminus \{M_i\} \text{ oraz } sInM := sInM \cup \{M_i\}$$

(2.11)

Rodzaje komunikacji

Kanały o niezerowej pojemności umożliwiają realizację operacji następujących typów komunikacji:

- **nieblokowanej**
- **blokowanej**

Komunikacja synchroniczna

W komunikacji synchronicznej, nadawca i odbiorca są blokowani aż odpowiedni odbiorca odczyta przesłaną do niego wiadomość (ang. *rendez-vous*).

Komunikacja asynchroniczna

W przypadku komunikacji asynchronicznej, nadawca lub odbiorca komunikuje się w sposób nieblokowany.

Stan procesu (1)

Stan $S_i(t)$ procesu w chwili t czasu lokalnego jest w ogólności zbiorem wartości wszystkich zmiennych lokalnych skojarzonych z procesem w chwili t oraz ciągów wiadomości wysłanych (wpisanych) do incydentnych kanałów wyjściowych i ciągów wiadomości odebranych z incydentnych kanałów wejściowych do chwili t.

Stan procesu (2)

Dla każdego t, $S_i(t) \in \mathcal{S}_i$. W celu uproszczenia notacji, zależność stanu od czasu można przyjąć za domyślną i jeśli nie prowadzi to do niejednoznaczności, oznaczać stan w pewnej chwili t przez S_i.

Zbiór $\mathcal{S}_i^0$ jest **zbiorem stanów początkowych**, których wartości są zadawane wstępnie, bądź są wynikiem zajścia wyróżnionego **zdarzenia inicjującego** E_i^0.

Definicja zdarzenia

Zdarzenie E_i^k odpowiada unikalnemu **wykonaniu operacji atomowej**, zmieniającemu stan S_i procesu i ewentualnie stan incydentnych z procesem kanałów $C_{i,j}$ lub $C_{j,i}$.

Jeżeli operacja odpowiadająca zdarzeniu została wykonana, to powiemy, że **zdarzenie zaszło**.

Klasy zdarzeń

- ***e_send***
- ***e_receive***
- ***e_internal***

Zdarzenie *e_send*

e_send (P_i, P_j, M)

zachodzi w procesie P_i, w wyniku wykonania przez ten proces operacji ***send***(P_i, P_j, M)

e_send $(P_i, \mathcal{P}_i^R, M)$

zachodzi w procesie P_i, w wyniku wykonania przez ten proces operacji ***send***$(P_i, \mathcal{P}_i^R, M)$

Zdarzenie *e_receive*

***e_receive* (P_i, P_j, M)**

zachodzi w procesie P_j, gdy P_j wykonał operację ***receive***(P_i, P_j, inM), a odczytana do zmiennej lokalnej inM wiadomość M pochodziła od procesu P_i

***e_receive* ($\mathcal{P}_j^S$, P_j, $\mathcal{M}_j^S$)**

zachodzi w procesie P_j, gdy P_j wykonał operację ***receive***($\mathcal{P}_j^S$, P_j, $sInM$), a odczytane do zmiennej lokalnej $sInM$ wiadomości $M_i \in \mathcal{M}_j^S$ pochodzą od procesów $P_i \in \mathcal{P}_j^S$

Zdarzenie *e_internal*

***e_internal* (P_i)**

zachodzi gdy proces P_i wykonał operację, która nie zmienia stanu jego kanałów incydentnych.

Do zdarzeń lokalnych zalicza się między innymi zdarzenia:

$e\_init(P_i, S_i^k)$ – które nadaje procesowi P_j stan S_i^k (w szczególności stan początkowy)

$e\_stop(P_i)$ – które kończy wykonywanie procesu

Dostępność wiadomości

Dostępność wiadomości utożsamiać można z zajściem zdarzeń w środowisku komunikacyjnym:

- zdarzenie dostarczenia wiadomości M $\boldsymbol{e\_deliver(P_i, P_j, M)}$
- zdarzenia nadejścia wiadomości M $\boldsymbol{e\_arrive(P_i, P_j, M)}$

Przez $\mathcal{P}_j^A$ oznaczać będziemy zbiór procesów, których wiadomości dotarły i są dostępne dla P_j.

Funkcja tranzycji

Funkcja tranzycji $\mathcal{F}_i \subseteq \mathcal{S}_i \times \mathcal{E}_i \times \mathcal{S}_i$ opisuje reguły zmiany stanu S na S' w wyniku zajścia zdarzenia E.

Elementy $\langle S, E, S_2 \rangle \in \mathcal{F}_i$ nazwiemy **tranzycjami** lub **krokami**. W zależności od zachodzącego zdarzenia E, tranzycję nazwiemy odpowiednio **tranzycją wejścia, wyjścia** lub **lokalną**.

Zdarzenia dopuszczalne

Funkcja tranzycji dopuszcza możliwość zajścia zdarzenia E tylko w tych stanach S, dla których $\langle S, E, S_2 \rangle \in \mathcal{F}_i$.

Dlatego też, w wypadku gdy $\langle S, E, S_2 \rangle \in \mathcal{F}_i$, powiemy że zdarzenie jest **dopuszczalne** (ang. *allowed*) w stanie S.

Wprowadzimy też predykat $allowed(E)$ oznaczający, że w danej chwili zdarzenie E jest dopuszczalne.

Zdarzenia gotowe

Oprócz czynnika wewnętrznego (stanu procesu), zajście zdarzenia może być dodatkowo uwarunkowane stanem kanałów wejściowych (środowiska). Jeśli zdarzenie może zajść ze względu na warunki zewnętrzne (stan kanałów), to powiemy że zdarzenie jest **przygotowane** lub **gotowe** (ang. *ready*).

Fakt gotowości zdarzenia E w danej chwili wyrażać będzie predykat $ready(E)$.

Predykat *enable*

Predykat $enable(E)$, oznacza, że zdarzenie jest **aktywne**, czyli jednocześnie **gotowe i dopuszczalne**. Stąd też:

$$enable(E) \equiv ready(E) \wedge allowed(E)$$

(2.14)

Procesy zakończone, wstrzymane

Powiemy, że proces P_i jest w **stanie końcowym** S_i^e, jeżeli zbiór zdarzeń dopuszczalnych w tym stanie jest pusty.

Jeżeli natomiast niepusty zbiór zdarzeń dopuszczalnych zawiera wyłącznie zdarzenia odbioru i żadne z tych zdarzeń nie jest aktywne (gotowe), to powiemy że proces jest **wstrzymany** (**zablokowany**).

Procesy aktywne, pasywne

Proces wstrzymany lub zakończony nazwiemy ***pasywnym***. Przez proces ***aktywny*** będziemy natomiast rozumieć proces, który nie jest pasywny.

Przyjmujemy, że w każdej chwili t stan procesu P_i reprezentuje zmienna logiczna $\boldsymbol{passive_i}$, przyjmująca wartość

- $True$, gdy proces P_i jest pasywny
- $False$, gdy proces P_i jest aktywny

Proces aktywny

Aktywny proces P_i ($passive_i$=$False$) może wysyłać i odbierać wiadomości, wykonywać tranzycje lokalne, a więc potencjalnie może również spontanicznie (w dowolnej chwili) zmienić swój stan na pasywny.

Proces pasywny

W stanie **pasywnym** procesu P_i ($passive_i$=$True$) dopuszczalne są natomiast co najwyżej zdarzenia odbioru.

Zmiana stanu procesu z pasywnego na aktywny uwarunkowana jest osiągnięciem gotowości przez choćby jedno z dopuszczalnych zdarzeń odbioru, czyli spełnieniem tak zwanego **warunku uaktywnienia**.

Warunek uaktywnienia

Warunek uaktywnienia (ang. *activation condition*) procesu P_i związany jest ze zbiorem warunkującym $\mathcal{D}_i$, zbiorem $\mathcal{P}_i^A$, oraz predykatem $activate_i(\mathcal{X})$.

Zbiór warunkujący

Zbiór warunkujący (ang. *dependent set*), jest sumą mnogościową zbiorów $\mathcal{P}_i^S$ wszystkich zdarzeń odbioru dopuszczalnych w danej chwili.

Predykat *activate*

Predykat $\boldsymbol{activate_i(\mathcal{X})}$ zdefiniowany jest w sposób następujący:

1. jeżeli $\mathcal{X} = \mathcal{D}_i$, to $activate_i(\mathcal{X}) = True$
2. jeżeli $\mathcal{X} = \varnothing$, to $activate_i(\mathcal{X}) = False$
3. jeżeli $\mathcal{X} \subset \mathcal{D}_i$ i $\mathcal{X} \neq \varnothing$, to:

$$activate_i(\mathcal{X}) \equiv \exists \mathcal{X}' :: \mathcal{X}' \neq \varnothing \wedge \mathcal{X}' \subseteq \mathcal{X} \wedge (\mathcal{P}_i^A = \mathcal{X}' \Rightarrow (passive_i \leftrightsquigarrow \neg passive_i)) \quad (2.15)$$

gdzie $passive_i \leftrightsquigarrow \neg passive_i$ oznacza, że pasywny proces P_i zmieni swój stan na aktywny w skończonym, choć nieprzewidywalnym czasie.

Predykat *ready*

Warunek uaktywnienia procesu P_i formalnie wyraża predykat $ready_i(\mathcal{X})$:

$$ready_i(\mathcal{X}) \equiv (\mathcal{P}_i^A \supseteq \mathcal{X}) \wedge activate_i(\mathcal{X}) \qquad (2.17)$$

Gdy proces jest uaktywniany, to wiadomości, których dostarczanie doprowadziło do spełnienia warunku uaktywnienia, są atomowo pobierane z buforów wejściowych i dalej przetwarzane.

Modele żądań

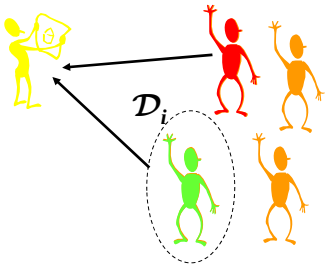
- model *jednostkowy*
- model *AND*
- model *OR*
- podstawowy model *k spośród r*
- model *OR-AND*
- dysjunkcyjny model *k spośród r*
- model *predykatowy*

Model *jednostkowy*

W modelu *jednostkowym* warunkiem uaktywnienia pasywnego procesu jest **przybycie wiadomości od jednego, ściśle określonego nadawcy**.

W tym przypadku $|\mathcal{D}_i| = 1$, dla każdego naturalnego i, $1 \leqslant i \leqslant n$.

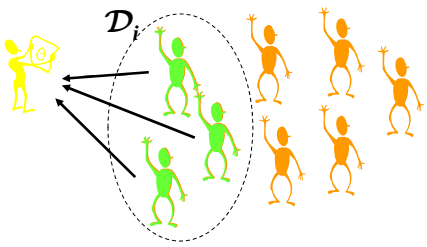
Model ten odpowiada szerokiej klasie systemów, w których procesy żądają kolejno po jednym tylko zasobie.



Model *AND*

W modelu *AND* proces pasywny staje się aktywnym, jeżeli dotarły wiadomości od wszystkich procesów tworzących zbiór warunkujący.

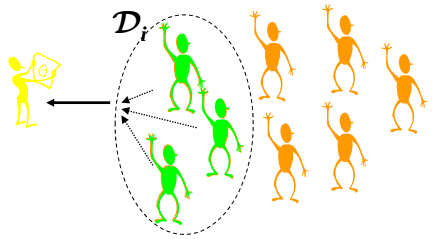
Model ten nazywany jest również **modelem zasobowym**.



Model *OR*

W modelu *OR* do uaktywnienia procesu wystarczy jedna wiadomość od któregokolwiek z procesów ze zbioru warunkującego.

Model ten nazywany jest również modelem *komunikacyjnym*.



(81)

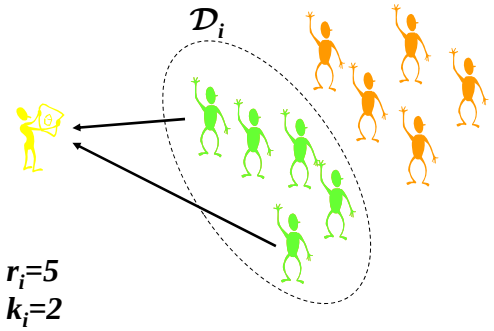
Podstawowy model *k spośród r*

W podstawowym modelu *k spośród r*, z pasywnym procesem P_i skojarzony jest zbiór warunkujący $\mathcal{D}_i$, liczba naturalna k_i, $1 \leqslant k_i \leqslant |\mathcal{D}_i|$, oraz liczba $r_i = |\mathcal{D}_i|$.

W modelu tym proces staje się aktywny tylko wówczas, gdy uzyska wiadomości od co najmniej k_i różnych procesów ze zbioru warunkującego $\mathcal{D}_i$.

(82)

Podstawowy model *k spośród r – przykład*

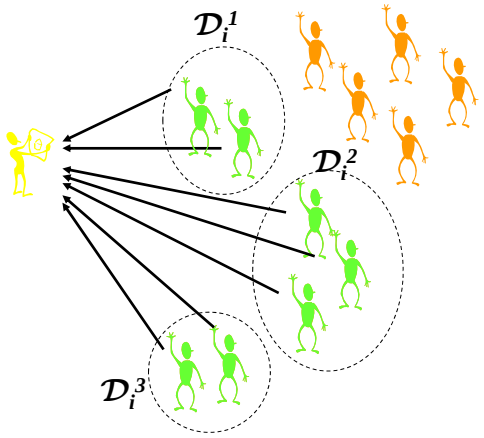


Model *OR-AND*

W modelu *OR-AND* zbiór warunkujący $\mathcal{D}_i$ pasywnego procesu P_i jest zdefiniowany jako $\mathcal{D}_i^1 \cup \mathcal{D}_i^2 \cup ... \cup \mathcal{D}_i^{q_i}$, gdzie dla każdego naturalnego u, $1 \leqslant u \leqslant q_i$, $\mathcal{D}_i^u \subseteq \mathcal{P}$.

Proces staje się aktywny po otrzymaniu wiadomości od każdego z procesów tworzących zbiór $\mathcal{D}_i^1$ lub od każdego z procesów tworzących zbiór $\mathcal{D}_i^2$ lub ... lub od każdego z procesów tworzących zbiór $\mathcal{D}_i^{q_i}$.

Model *OR-AND* – przykład



Model *predykatowy*

W modelu *predykatowym*, dla każdego pasywnego procesu P_i ze zbiorem warunkującym $\mathcal{D}_i$ określony jest predykat $activate_i(\mathcal{X})$, gdzie $\mathcal{X} \subseteq \mathcal{P}$.

Jak łatwo zauważyć, stosownie definiując predykat $activate_i(\mathcal{X})$ można oczywiście uzyskać wszystkie wcześniej omówione modele żądań.

Relacja poprzedzania zdarzeń (1)

Oznaczmy przez $\mapsto$ **relację poprzedzania** (ang. *happen before, causal precedence, happened before*) zdefiniowaną na zbiorze Λ w następujący sposób:

$E_i^k \mapsto E_j^l \Leftrightarrow$

1) $i = j \ \wedge \ k < l$, lub
2) $i \neq j$ oraz E_i^k jest zdarzeniem $e\_send(P_i, P_j, M)$ wysłania wiadomości M, a zdarzenie E_j^l jest zdarzeniem $e\_receive(P_i, P_j, M)$ odbioru tej samej wiadomości, lub
3) istnieje sekwencja zdarzeń $E^0, E^1, E^2, \ldots, E^s$, taka że $E^0 = E_i^k$, $E^s = E_j^l$ i dla każdej pary $\langle E^u, E^{u+1} \rangle$ gdzie $0 \leqslant u \leqslant s$-1, zachodzi 1) albo 2).

Relacja poprzedzania lokalnego

Przez $\mapsto_i$ oznaczamy **relację poprzedzania lokalnego** zdarzeń procesu, taką że:

$$E_i^k \mapsto_i E_j^l$$

wtedy i tylko wtedy, gdy $i=j$ oraz $k<l$ (lub gdy $i=j$ oraz $E_i^k \mapsto E_j^l$).

Zdarzenia przyczynowo-zależne

Zdarzenia E_i^k i E_j^l nazywamy **przyczynowo-zależnymi**, jeżeli:

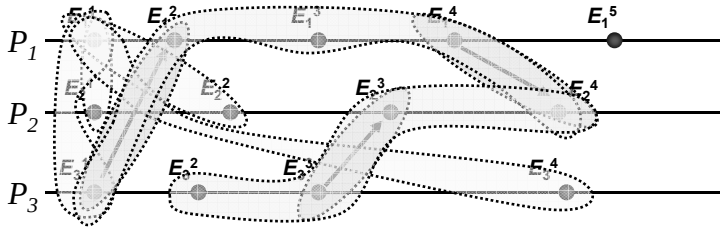
$$E_i^k \mapsto E_j^l \quad \text{albo} \quad E_j^l \mapsto E_i^k \tag{3.9}$$

W przeciwnym razie zdarzenie te nazwiemy **przyczynowo-niezależnymi** lub współbieżnymi (ang. *concurrent*, *causally independent*), co będziemy oznaczać przez $E_i^k \parallel E_j^l$.

Diagramy przestrzenno-czasowe

Realizację przetwarzania rozproszonego można przedstawić graficznie w postaci **diagramu przestrzenno-czasowego** (ang. *space-time diagram*), w którym osie reprezentują upływ czasu globalnego, a punkty na osiach **–** zdarzenia.

Przykładowy diagram



$E_1^1 || E_2^1$, $E_1^1 || E_2^2$, $E_1^1 || E_3^1$, $E_1^1 || E_3^4$,
$E_3^1 \mapsto E_1^2$, $E_3^3 \mapsto E_2^3$, $E_1^4 \mapsto E_2^4$,
$E_3^1 \mapsto E_1^4$, $E_3^1 \mapsto E_2^4$, $E_3^2 \mapsto E_2^4$

Relacja poprzedzania stanów lokalnych

Przez analogię do relacji na zbiorze zdarzeń, można zdefiniować częściowy porządek na zbiorze stanów wszystkich procesów $P_i \in \mathcal{P}$ w sposób następujący:

$$S_i^k \mapsto S_j^l \Leftrightarrow \begin{cases} E_i^{k+1} \mapsto E_j^l, \text{ lub} \\ E_i^{k+1} = E_j^l \end{cases} \quad (3.10)$$

Stany współbieżne

Stany lokalne, dla których nie zachodzi ani relacja $S_i^k \mapsto S_j^l$ ani też relacja $S_j^l \mapsto S_i^k$, nazywamy **współbieżnymi**.

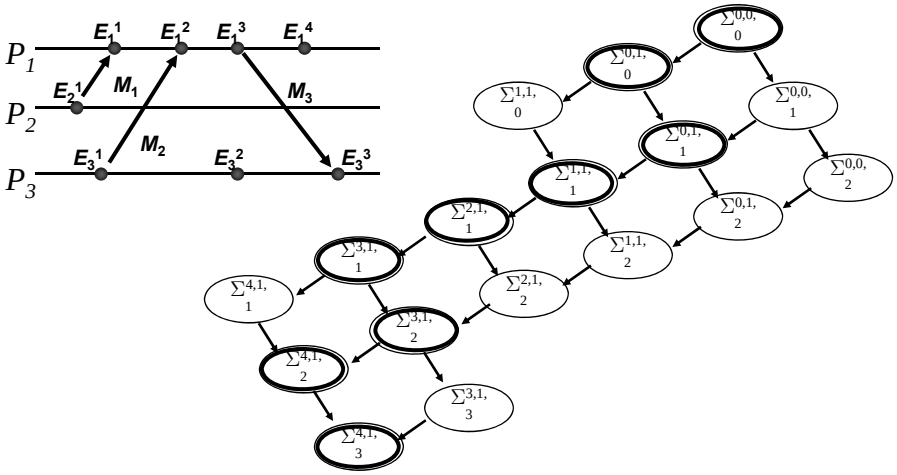
Graf stanów osiągalnych

Zbiór częściowo uporządkowany $\langle \boldsymbol{\Lambda}, \mapsto \rangle$ może być przedstawiony w postaci grafu zorientowanego, w którym wierzchołki odpowiadają stanom Σ, a łuki $\langle \Sigma^k, \Sigma^l \rangle$ oznaczają istnienie zdarzenia dopuszczalnego takiego, że

$$\langle \Sigma^k, E, \Sigma^l \rangle \in \boldsymbol{\Phi}. \qquad (3.11)$$

Graf taki, będziemy nazywać **grafem stanów osiągalnych przetwarzania rozproszonego** lub **siatką obliczeń rozproszonych**.

Przykład grafu stanów osiągalnych



Niedeterminizm przetwarzania

W kontekście grafu stanów osiągalnych przetwarzania rozproszonego, każda realizacja przetwarzania rozproszonego jest pewną ścieżką w tym grafie. Istnienie wiele różnych ścieżek ilustruje **niedeterminizm** przetwarzania rozproszonego, oznaczający że dla danego stanu może istnieć wiele stanów następnych.

Niedeterministyczne zdarzenie lokalne

Lokalne zdarzenie procesu jest **niedeterministyczne**, gdy jego zajście może być zastąpione przez zajście innego zdarzenia i wybór ten nie jest przewidywalny.

Jeżeli przykładowo sekwencyjne wykonanie procesu może być w każdej chwili zmienione w wyniku zajścia przerwania zewnętrznego, to wszystkie zdarzenia tego procesu są niezdeterminowane.

Przetwarzanie zdeterminowane i niedeterministyczne

Przetwarzanie nazywamy **zdeterminowanym** jeżeli wszystkie zdarzenia są zdeterminowane. W przeciwnym wypadku, przetwarzanie nazywamy **niedeterministycznym**.

Przetwarzanie quasi-deterministyczne

W ramach niedeterministycznego przetwarzania rozproszonego wyróżnia się podklasę przetwarzania **quasi deterministycznego** (ang. *quasi-deterministic*, *piece-wice deterministic*, *event-driven*), w której niedeterminizm jest wyłącznie konsekwencją niedeterminizmu operacji odbioru.

Diagramy równoważne

Należy zauważyć, że w ogólności istnieje wiele różnych diagramów przestrzenno-czasowych, którym odpowiada taki sam zbiór częściowo uporządkowany $\langle \Lambda, \mapsto \rangle$. Diagramy takie nazywa się **diagramami równoważnymi**.

Przykład diagramów równoważnych