Programowanie współbieżne i rozproszone

WYKŁAD 1

Jan Kazimirski

Wymagania

- Znajomość podstawowych zagadnień związanych z architekturą komputerów i systemami operacyjnymi.
- Podstawy obsługi systemu operacyjnego Linux.
- Programowanie C/C++ w środowisku Linux.
- Programowanie PHP.

Opis zajęć

- Podstawy programowania współbieżnego i rozproszonego
- Programowanie klastrowe – MPI
- Obliczenia na GPU – CUDA i Thrust
- Programowanie wielowątkowe w środowisku Linux
- Usługi rozproszone (SOAP)

Podstawy programowania współbieżnego i rozproszonego

- Ogólne spojrzenie na rozwój wymagań wobec mocy obliczeniowej komputerów
- Wykorzystanie różnych modeli współbieżności we współczesnych systemach komputerowych
- Zagadnienia i wyzwania związane z tworzeniem programów współbieżnych

Programowanie klastrowe – MPI

- Klaster komputerów jest ekonomicznym sposobem na zwiększenie wydajności lub/i niezawodności systemu komputerowego
- Biblioteka MPI pozwala na proste tworzenie aplikacji działających w środowisku klastra
- Funkcje biblioteki MPI ukrywają szczegóły komunikacji między węzłami i topologię klastra

Obliczenia na GPU – CUDA i Thrust

- Współczesne karty graficzne oferują bardzo wysoką wydajność obliczeniową
- API dostarczane przez producentów kart pozwala wykorzystać układy GPU do obliczeń
- Specyfika sprzętowa GPU wymaga specyficznych algorytmów
- Biblioteka CUDA pozwala na dostęp do mocy obliczeniowej kart nVidia z poziomu programu C/C++.
- Biblioteka Thrust udostępnia wysokopoziomowy, zorientowany obiektowo interfejs do obsługi GPU

Programowanie wielowątkowe w środowisku Linux

- Powszechne stosowanie procesorów wielordzeniowych pozwala na łatwe zwiększenie wydajności poprzez równoległe wykorzystanie kilku rdzeni.
- Mechanizm wątków pozwala na jednoczesne wykorzystanie wielu rdzeni procesora. Mechanizm ten jest dostępny we wszystkich współczesnych systemach operacyjnych
- Biblioteka PThreads pozwala na łatwe tworzenie programów wielowątkowych.

Usługi rozproszone (SOAP)

- Współczesne systemy komputerowe często wymagają komunikacji z innymi systemami.
- Model usług rozproszonych zakłada współdziałanie wielu różnych systemów za pomocą standardowych protokołów
- Często stosowanym protokołem takiej komunikacji jest SOAP (Simple Object Access Protocol)
- Język PHP pozwała łatwo implementować usługi SOAP

Zaliczenie końcowe

- Warunki zaliczenia
  - Obecność na zajęciach ***75%***
  - Projekt zaliczeniowy ***25%***
- Ocena końcowa
  - ***50%-74%*** ocena **dostateczna**
  - ***75%-90%*** ocena **dobra**
  - ***91%-100%*** ocena **bardzo dobra**

Przykłady projektu zaliczeniowego

- *Znaleźć 5-cio cyfrowe liczby całkowite których suma silni z poszczególnych cyfr jest równa początkowej liczbie.*
- *Znaleźć wszystkie 6-cyfrowe liczby pierwsze takie, w których kolejne cyfry są większe od poprzedniej, oraz suma cyfr jest również liczba pierwsza.*
- *Znaleźć wszystkie czwórki 2-cyfrowych liczb całkowitych takich, ze suma ich sześcianów wynosi dokładnie 1 000 000*
- Osoby zainteresowane wykonaniem projektu proszę o kontakt mailem.
- Program musi być napisany w języku C/C++ i wykorzystywać bibliotekę MPI lub CUDA/Thrust

Komputery równoległe

Wydajność komputerów

- Rozwój technologii wiąże się z ciągłym wzrostem wydajności komputerów
- Pierwsze komputery – 1-100 operacji/sek.
- Najszybszy obecnie superkomputer – **Sequoia (IBM)**
  - 16,32 PFLOPS (peta - 10^{15})
  - 1769472 rdzeni
  - Pamięć operacyjna: 1572864 GB

Granice wydajności komputerów

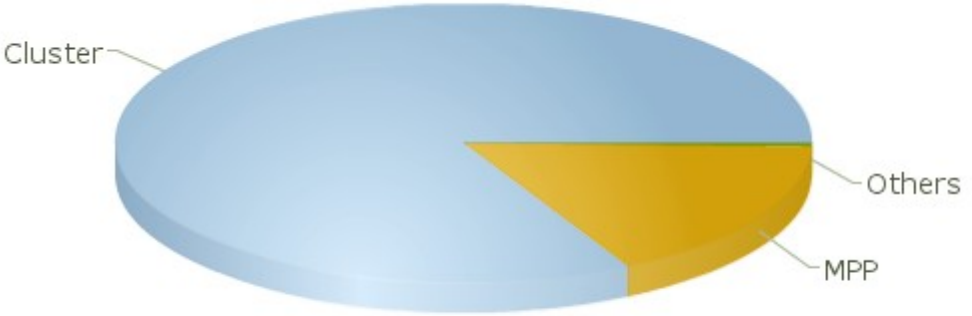
- Szybkość zegara
  - Ograniczenia wynikające z teorii względności
  - Taktowanie 10 Ghz – długość ścieżki < 2cm
  - W przypadku zegara 1 Thz – komputer musi mieć wielkość nie więcej niż 100 mikrometrów.
- Rozpraszanie ciepła
  - Im mniejszy komputer tym trudniej odprowadzić ciepło.

Współczesne superkomputery

- Lista rekordzistów – www.top500.org
- Na stronie znajduje się ranking najszybszych komputerów świata.
- Lista aktualizowana jest 2 razy do roku.
- Ostatnie rekordy:
  - 06/2012 Sequoia 16,32 PFLOPS
  - 11/2011 K computer 10,51 PFLOPS
  - 11/2010 Tianhe-I 2,56 PFLOPS

Top500

Systemy wieloprocesorowe i rozproszone dominują na liście Top500

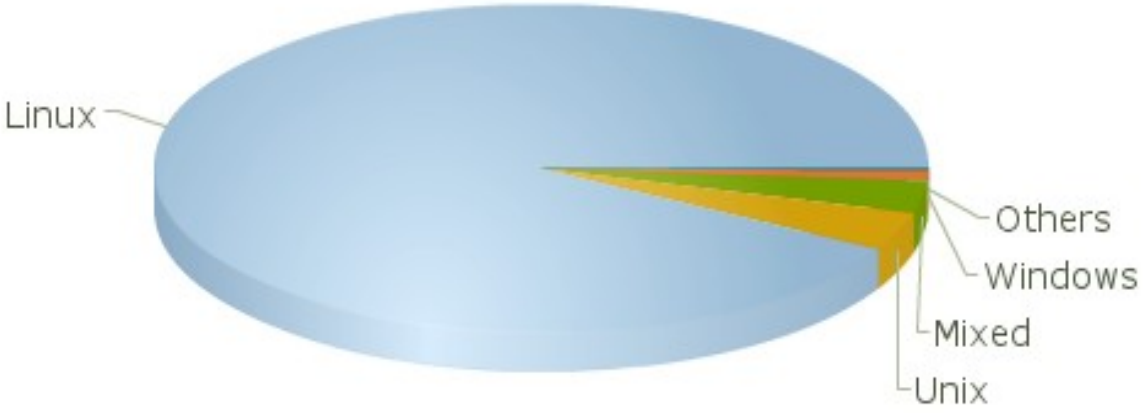


Źródło: www.top500.org

Top500 c.d.

DYGRESJA

Czy ktoś jeszcze uważa, że nie warto uczyć się Linuksa???



Wektoryzacja na poziomie procesora

- Klasyczny komputer – architektura szeregowa
- Wzrost taktowania nie jest wystarczający do efektywnego wzrostu wydajności
- Techniki wektoryzacji na poziomie procesora – potoki, jednostki superskalarne, out-of-order execution

Potoki

- „Linia montażowa” procesora
- Rozkaz podzielony na fazy (pobranie, dekodowanie, pobranie operandów, wykonanie itp.)
- Poszczególne podzespoły jednostki wykonawczej wykonują poszczególne fazy rozkazu.
- Kolejny rozkaz może być pobrany zanim skończy się wykonywanie rozkazów poprzednich
- Pentium4 – potoki 20 fazowe.

Superskalarność

- Zwielokrotnienie liczby jednostek wykonawczych.
- Przykład: PowerPC 970
  - 4 jednostki ALU
  - 2 jednostki FPU
  - 2 jednostki SIMD
- Procesor może wykonywać kilka rozkazów jednocześnie.
- Zwykle łączona jest z potokami.

Inne techniki

- Out-of-order execution – zmiana kolejności wykonywania rozkazów w celu eliminacji zależności i lepszego wykorzystania jednostek wykonawczych
- Wykonywanie spekulatywne – wykonywanie instrukcji za skokiem warunkowym (z wyprzedzeniem).

Taksonomia Flynna

- Klasyfikacja architektur komputerów ze względu na sposób przetwarzania:
  - **SISD** – Single Instruction, Single Data – komputer skalarny
  - **SIMD** – Single Instruction, Multiple Data – komputer wektorowy
  - **MISD** – Multiple Instruction, Single Data
  - **MIMD** – Multiple Instruction, Multiple Data – system wieloprocesorowy, klaster

Model pamięci

- Pamięć dzielona – pamięć wspólna dla wszystkich jednostek wykonawczych
- Pamięć rozproszona – poszczególne jednostki mają osobne obszary pamięci
- Architektury UMA/NUMA
  - UMA – Uniform Memory Access
  - NUMA – Non-Uniform Memory Access

Komputery równoległe

- Procesory wielordzeniowe
- Symetryczna multiprocesorowość (SMP)
- System wieloprocesorowy z rozproszoną pamięcią
- Klastry komputerowe
- MPP – masowa równoległość
- Środowiska gridowe

Oprogramowanie

- Rozproszone systemy operacyjne
  - Ukrywanie szczegółów architektury przed użytkownikiem końcowym
- Aplikacje równoległe
  - **Jawne** – pełna kontrola programisty.
  - **Częściowo jawne** – pod kontrolą programisty i kompilatora
  - **Niejawne** – pod całkowitą kontrolą kompilatora

Beowulf

- Klaster komputerowy dający dużą wydajność niewielkim kosztem
- Budowany ze standardowych komputerów PC połączonych siecią Ethernet
- Zazwyczaj pod kontrolą systemu Linux
- Wykorzystuje biblioteki MPI lub PVM
- Często wykorzystywany do obliczeń naukowych

Mosix

- Oparty na jądrze Linuksa
- Realizuje model SSI (Single System Image) użytkownik widzi klaster jako jedną maszynę
- Zawiera mechanizmy równoważenia obciążenia – potrafi przenosić zadania pomiędzy węzłami (migracja zadań)

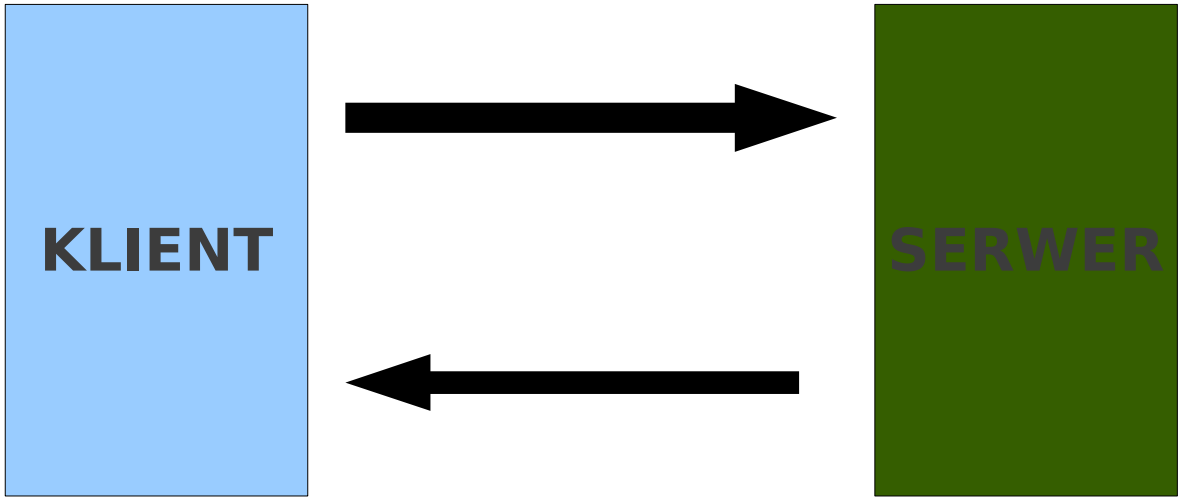
Rozproszony system plików

- Umożliwia połączenie przestrzeni dyskowych wielu komputerów (klaster)
- Dane są dostępne dla dowolnego węzła w klastrze w sposób przezroczysty
- Przykład: system plików **Lustre** (stosowany m.in. w klastrach Tianhe-1A i Jaguar

Technologia klient/serwer

- Klient – komputer osobisty lub stacja robocza. Środowisko graficzne, wygodne w użyciu.
- Serwer – wydajny system z zestawem usług dla klientów. Usługi mogą być współdzielone – np. serwer bazodanowy obsługujący pracowników firmy

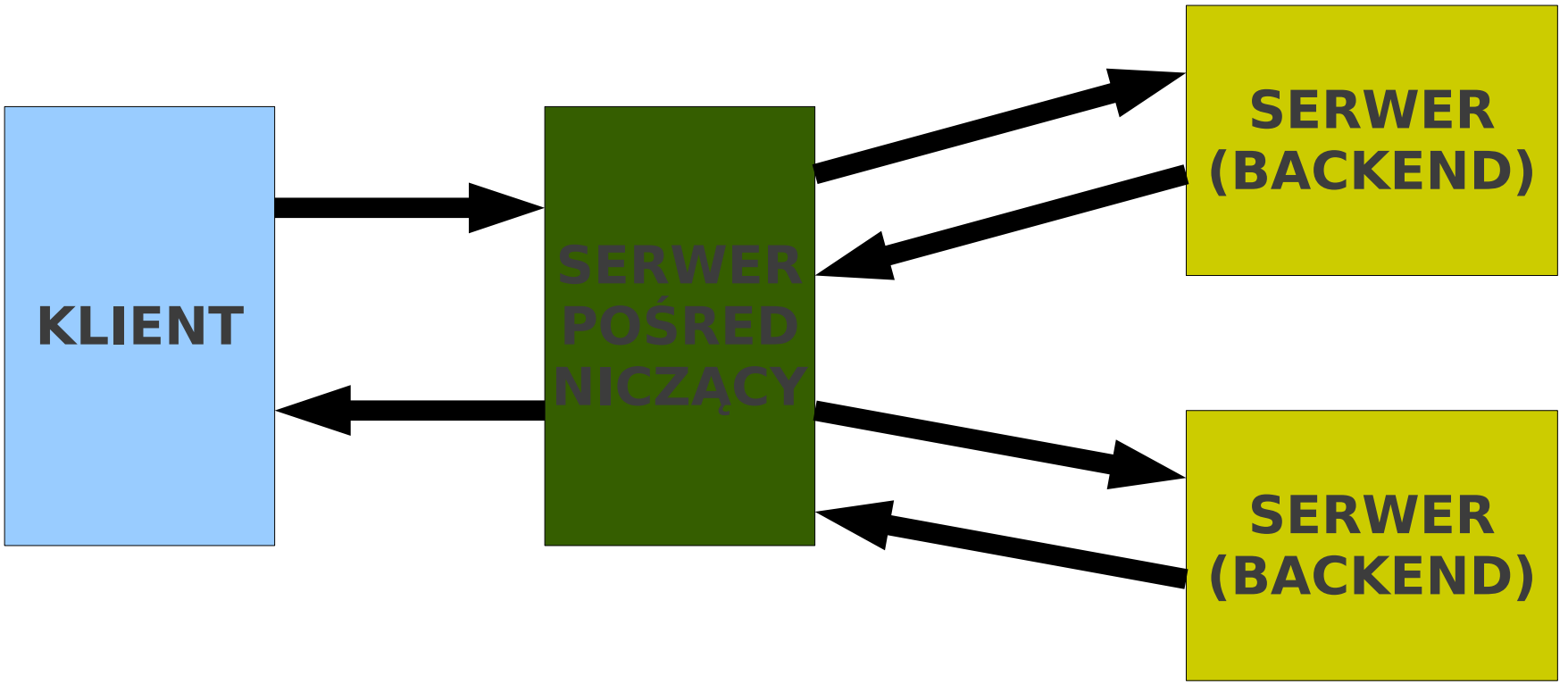
Architektura 2-warstwowa



Klasy aplikacji klient serwer

- Przetwarzanie na głównym komputerze
  - Większość pracy odbywa się na serwerze. Klient jest tylko terminalem
- Przetwarzanie po stronie serwera („cienki klient”)
  - Dane przetwarzane są na serwerze. Klient odpowiada za ich prezentację
- Przetwarzanie po stronie klienta („gruby klient”)
  - Serwer dostarcza surowe dane. Przetwarzanie odbywa się po stronie klienta
- Przetwarzanie zespołowe
  - Przetwarzanie rozłożone pomiędzy serwer i klienta

Architektura 3-warstwowa



Zdalne wywoływanie procedur (RPC)

- Komunikacja programów na różnych komputerach za pomocą wywoływania procedur w sposób zdalny
- Prosta składnia.
- Przejrzystość. Wywołanie niewiele różni się od wywołania lokalnego
- Przejrzyste, przenośne interfejsy.

Powiązanie klienta i serwera

- Powiązanie nietrwałe – połączenie zestawiane tylko na czas wykonania procedury.
- Powiązanie trwałe – zestawiane raz i podtrzymywane. Może być wykorzystane ponownie do wywoływania tej samej lub innych zdalnych procedur.

Synchroniczne i asynchroniczne RPC

- Synchroniczne RPC – po wywołaniu procedury klient czeka na rezultat.
- Asynchroniczne RPC – nie blokuje klienta. Klient i serwer mogą niezależnie realizować zadania.

Mechanizmy zorientowane obiektowo

- Obiekty na zdalnych komputerach komunikują się poprzez wymianę komunikatów.
- Klient wysyła żądanie do obiektu brokera (katalog dostępnych usług)
- Broker wywołuje odpowiedni obiekt i przekazuje mu dane.
- Odpowiedź zdalnego obiektu zwracana jest klientowi.

Mechanizmy zorientowane obiektowo c.d.

- **COM** – Component Object Model (Microsoft)
- **CORBA** – Common Object Request Broker Architecture (IBM, Apple, Sun)
- **DCOP** – Desktop Communication Protocol (stosowany w KDE)
- **SOAP** – Simple Object Access Protocol (standard W3C – korzysta z XML i zwykle protokołu HTTP).

Grid

- System integrujący dużą liczbę urządzeń znajdujących się w różnych lokalizacjach (często odległych)
- Obejmuje komputery, infrastrukturę sieciową, nośniki danych oraz różnorodne sensory.
- Widziany jako wirtualny superkomputer (przezroczysty dostęp do rozproszonych zasobów).
- Otwarte standardy – łączenie różnorodnych technologii.
- Architektura oparta o usługi.

SETI@home

- Jeden z pierwszych „nieoficjalnych” gridów
- Uruchomiony w 1999 roku w celu poszukiwania śladów obcych cywilizacji w kosmosie
- Ogólnie dostępny program klienta (Windows, Linux, Mac OS).
- Porcje danych wysyłane z serwera i przetwarzane przez klienta (screensaver)

SETI@home c.d.

- Liczba uczestników projektu (w całym okresie trwania) to ponad 5 mln.
- Projekt jest posiadaczem rekordu Guinnessa za największe obliczenia w historii.
- W 2009 roku moc obliczeniowa projektu wynosiła ponad 769 TFLOPS (RoadRunner – 1105 TFLOPS)

World Community Grid

- Oryginalny projekt SETI@home obejmował tylko jedno zagadnienie
- WCG jest publicznym środowiskiem gridowym służącym do różnorodnych obliczeń.
- Statystyki:
  - ponad 500 tys. użytkowników
  - ponad 1.5 mln komputerów
  - całkowity czas pracy – ponad 400 tys. lat

World Community Grid c.d.

- Projekty:
  - Computing for Clean Water
  - The Clean Energy Project
  - Help Cure Muscular Dystrophy
  - Help Fight Childhood Cancer
  - Help Conquer Cancer
  - Human Proteome Folding
  - FightAIDS@Home

World Community Grid

Welcome back **jankazim**

Registered Member Since: 08-07-12 08:03:46

My Statistics	
Total Run Time (y:d:h:m:s) (Rank)	2:061:06:41:52 (#47,756)
Points Generated (Rank)	3,739,839 (#25,445)
Results Returned (Rank)	5,741 (#30,392)

My Team

World Community Grid has hundreds of teams where you can contribute as part of a university, country, city, website community, or many, many other themes. Joining a team does not affect your individual contribution, but it does allow you to participate as part of a larger group. To search for a team to join, click here.

Statistics By Projects

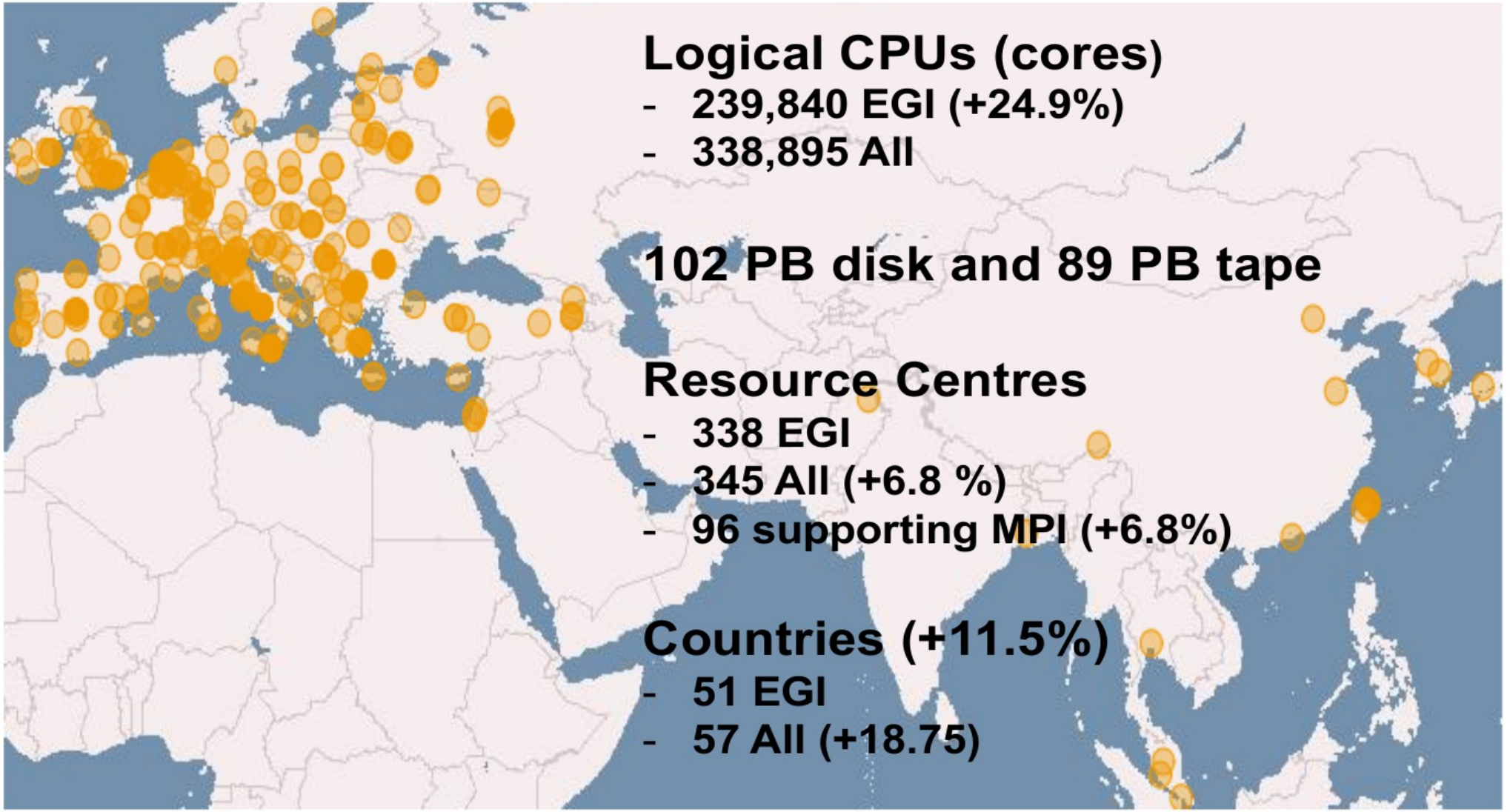
Statistics Last Updated: 13-03-02 00:06:02 (UTC) [12 hour(s) ago]

Project	Points Generated	Results Returned	Total Run Time (y:d:h:m:s)	Badges Earned
The Clean Energy Project - Phase 2	514	1	0:000:12:00:01	
Help Conquer Cancer	595,816	2,339	0:172:19:29:26	
Human Proteome Folding - Phase 2	533	1	0:000:18:46:12	
FightAIDS@Home	2,437,259	2,700	1:159:23:11:59	
Help Cure Muscular Dystrophy - Phase 2	700,705	690	0:083:20:35:33	
Discovering Dengue Drugs - Together	5,012	10	0:008:08:38:41	

EGI

- European Grid Initiative (EGI) – europejskie środowisko gridowe
- Status na 2010 r.
  - 10000 użytkowników
  - prawie 250 000 procesorów (rdzeni)
  - 40 petabajtów przestrzeni dyskowej
  - 317 węzłów w 52 krajach

EGI



Źródło: EGI-InSPIRE RI-261323

PL-Grid

- Polskie środowisko gridowe
- Uczestnicy
  - Cyfronet AGH (Kraków)
  - ICM (Warszawa)
  - Poznańskie Centrum Superkomputerowe
  - Akademickie Centrum Komputerowe, Gdańsk
  - Wrocławskie Centrum Superkomputerowe

PL-Grid c.d.

- Zasoby
  - moc obliczeniowa 230,16 TFlops
  - przestrzeń dyskowa 3,6 PB
  - Pamięć operacyjna 51,33 TB
  - Liczba rdzeni 23 616
- Obszary zastosowań: biologia, chemia, fizyka, symulacje numeryczne

Programowanie współbieżne

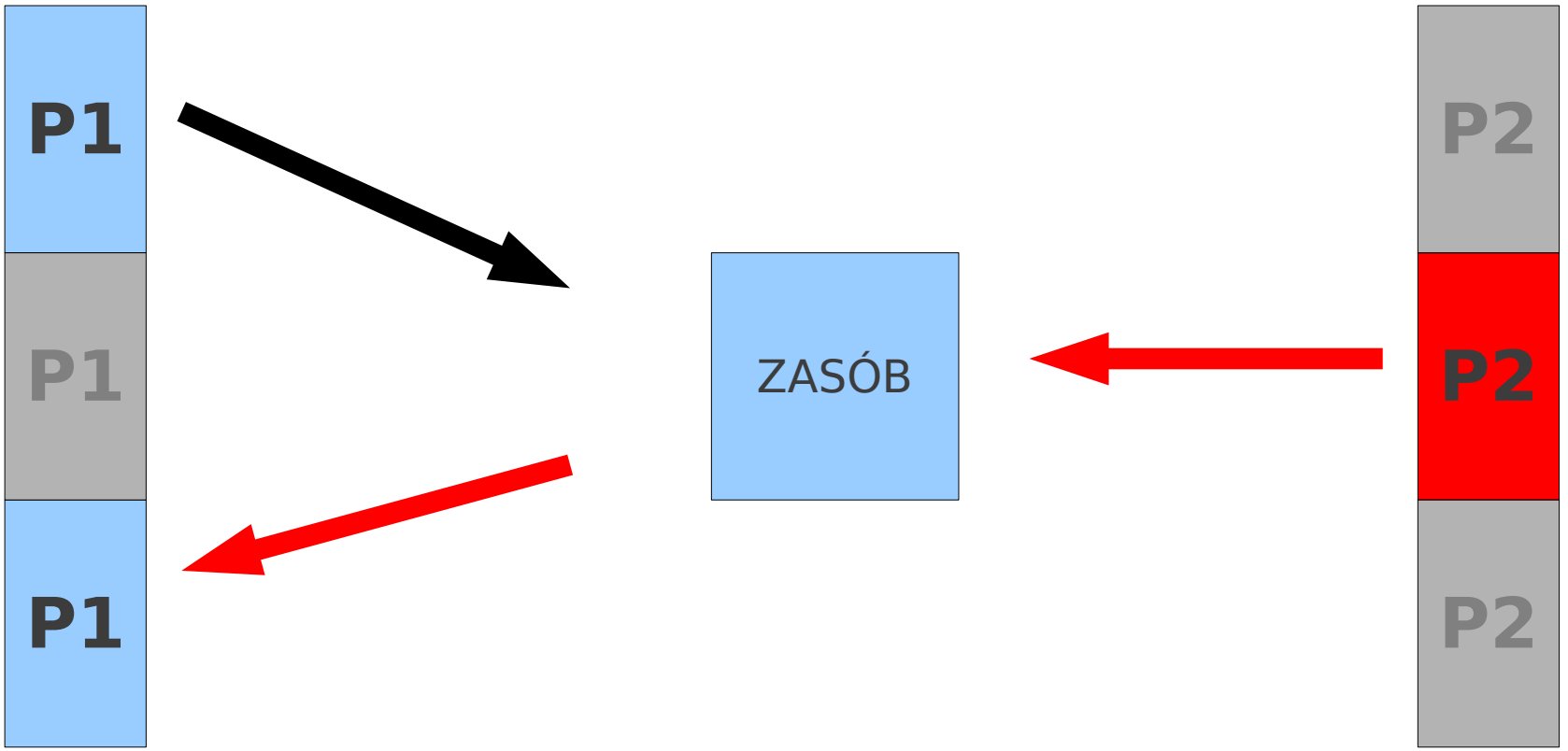
Współbieżność

- Wielozadaniowość
  - zarządzanie wieloma procesami w ramach jednego CPU
- Wieloprocesorowość
  - zarządzanie wieloma zadaniami w systemie z wieloma CPU
- Przetwarzanie rozproszone
  - zarządzanie wieloma zadaniami w architekturze rozproszonej

Problemy współbieżności

- Współdzielenie zasobów globalnych
  - Trudna do określenia kolejność dostępu do zasobów globalnych (np. zmienne).
- Optymalna alokacja zasobów dla procesów
  - Wydajne wykorzystywanie kanałów we/wy. Unikanie impasu.
- Problemy z testowaniem programów
  - Niedeterministyczne wykonywanie aplikacji wielowątkowych / wielozadaniowych.

Dostęp do zasobów – współbieżność



Współbieżność i zmienne globalne

```
Program:

void echo() {
    chin =
getchar();
    chout = chin;
    putchar(chout);
};
```

Zmienna globalna

Równoległe wykonanie echo()

P1	P2
chin = getchar();	-
-	chin = getchar();
-	chout = chin;
chout = chin;	-
putchar(chout);	-
-	putchar(chout);

Zasoby globalne w systemie wielozadaniowym

- Wymagają ochrony – zwykle przydzielane na zasadzie wyłączności
- Jeden proces korzysta z zasobu – pozostałe czekają uśpione na jego zwolnienie.
- **Problem** – system przerwań
- **Problem (system wieloprocesorowy)** – jednoczesny dostęp do zasobu.

Interakcja procesów

- Procesy w pełnej izolacji (nieświadome siebie nawzajem) – rywalizacja o zasoby
- Procesy pośrednio świadome siebie nawzajem – współpraca przez współdzielenie
- Procesy bezpośrednio świadome siebie nawzajem – możliwa komunikacja.

Rywalizacja o zasoby

- Dwa procesy próbują uzyskać dostęp do zasobu
- Procesy nie wiedzą o sobie, są od siebie odizolowane
- Przegrany proces zostanie spowolniony (musi czekać na zasób), być może nigdy go nie otrzyma.

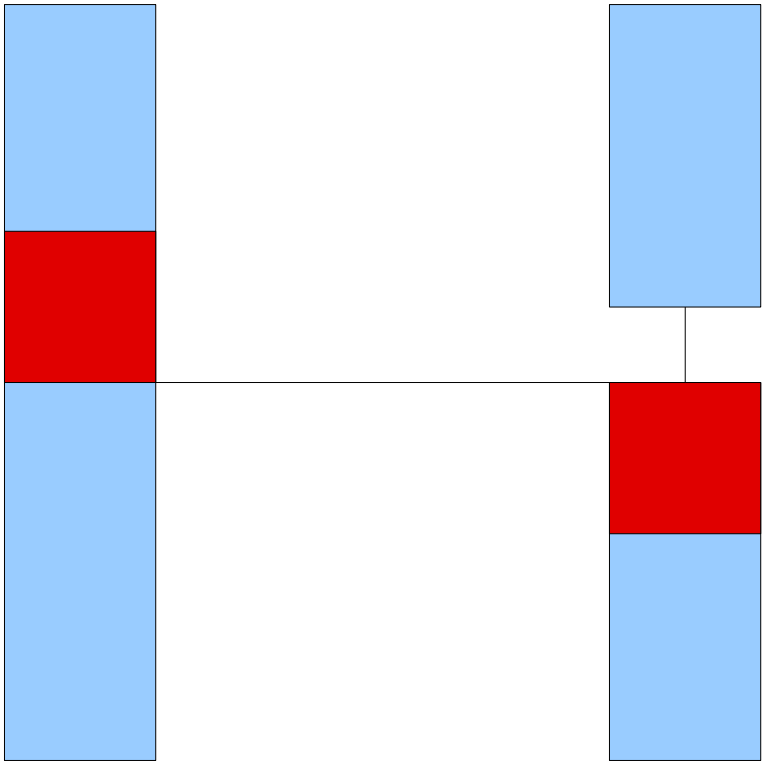
Wzajemne wykluczenie

- Kilka procesów próbuje uzyskać dostęp do zasobu – **rywalizacja o zasób**.
- Procesy są od siebie odizolowane, nic o sobie nie wiedzą.
- Tylko jeden z procesów otrzyma zasób, a pozostałe muszą czekać – **wzajemne wykluczenie**.

Wzajemne wykluczenie c.d.

- Niepodzielny zasób wymagający dostępu „na wyłączność” - **zasób krytyczny**.
- Fragment kodu realizujący dostęp do zasobu krytycznego – **sekcja krytyczna**.
- Mechanizm wzajemnego wykluczania zapewnia że tylko jeden z procesów wykonuje swoją sekcję krytyczną dla danego zasobu.

Sekcja krytyczna



Sekcja krytyczna c.d.

Program:

```
void echo() {
    chin =
getchar();
    chout = chin;
    putchar(chout);
};
```

Zmienna globalna

Równoległe wykonanie echo()

P1	P2
cs.lock() ;	-
chin = getchar();	-
-	cs.lock();
chout = chin;	-
putchar(chout);	-
cs.unlock();	-
-	chin = getchar();
-	chout = chin;
-	putchar(chout);
-	cs.unlock();

Impas

P1	P2
…	…
R1.lock();	…
…	**R2.lock();**
…	…
R2.lock();	**R1.lock();**

- Dwa procesy P1 i P2, oraz dwa zasoby R1 i R2.
- Każdy proces wymaga dostępu do obu zasobów
- P1 rezerwuje R1, a P2 rezerwuje R2
- Każdy proces czeka na zasób zarezerwowany przez drugi z procesów

Zagłodzenie

P1	P2	P3
…	…	…
R.lock();	**R.lock();**	**R.lock();**
…		
R.unlock();		
…	…	
R.lock();	…	
	R.unlock();	
…	…	
…	**R.lock();**	
R.unlock();		
…	…	

Procesy P1,P2,P3 wymagają dostępu do zasobu R

P1 posiada zasób, P2 i P3 czekają

P1 wychodzi z sekcji krytycznej, zasób otrzymuje P2.

P1 ponownie zgłasza zapotrzebowanie na zasób

P2 zwalnia zasób, system przydziela go do P1. P3 czeka

P3 może nigdy nie otrzymać zasobu, pomimo braku impasu

Algorytmy wzajemnego wykluczania - założenia

- Musi być wymuszone. Tylko jeden proces może być w swojej sekcji krytycznej (dla danego zasobu)
- Proces zatrzymany nie może przeszkadzać innym procesom
- Proces oczekujący w końcu musi otrzymać zasób
- Pierwszy proces żądający dostępu do swojej sekcji krytycznej otrzymuje go
- Mechanizm nie może zależeć od liczby i szybkości procesów
- Proces może przebywać w sekcji krytycznej tylko określony czas

Algorytm wzajemnego wykluczania – architektura z 1 procesorem

- W architekturze z jednym procesorem w danym momencie wykonywany jest tylko jeden proces.
- Wymuszenie wzajemnego wykluczania:
  - Blokada przełączania procesów w czasie wykonywania sekcji krytycznej
  - Wyłączenie przerwań na czas wykonywania sekcji krytycznej procesu

Algorytm wzajemnego wykluczania – architektura wieloprocesorowa

- W architekturze wieloprocesorowej procesy na różnych procesorach wykonują się jednocześnie.
- Zablokowanie przerwań i przełączania procesów w sekcji krytycznej nie rozwiązuje problemu
- Wykorzystywana jest własność pamięci operacyjnej – procesor ma wyłączny dostęp do danej komórki pamięci.
- Specjalne rozkazy procesora – „odczyt i zapis” lub „odczyt i test” realizowane jako operacje atomowe (nieprzerywalne)

Wzajemne wykluczenie poprzez zmienną współdzieloną

- Procesy P1,P2...,Pn wymagają dostępu do zasobu – kontrola za pomocą zmiennej T
- T=0, zasób wolny
- P1 testuje i ustawia T – rezerwuje zasób
- Pozostałe procesy czekają na zwolnienie zasobu (**aktywne oczekiwanie**)
- P1 zwalnia zasób, T->0, zasób przejmuje kolejny proces.

Specjalne rozkazy procesora – zalety i wady

- **Dowolna liczba procesów, systemy jedno- i wieloprocesorowe**
- **Prostota**
- **Obsługa wielu sekcji krytycznych**
- **Wykorzystanie aktywnego oczekiwania**
- **Możliwy stan zagłodzenia i impas.**

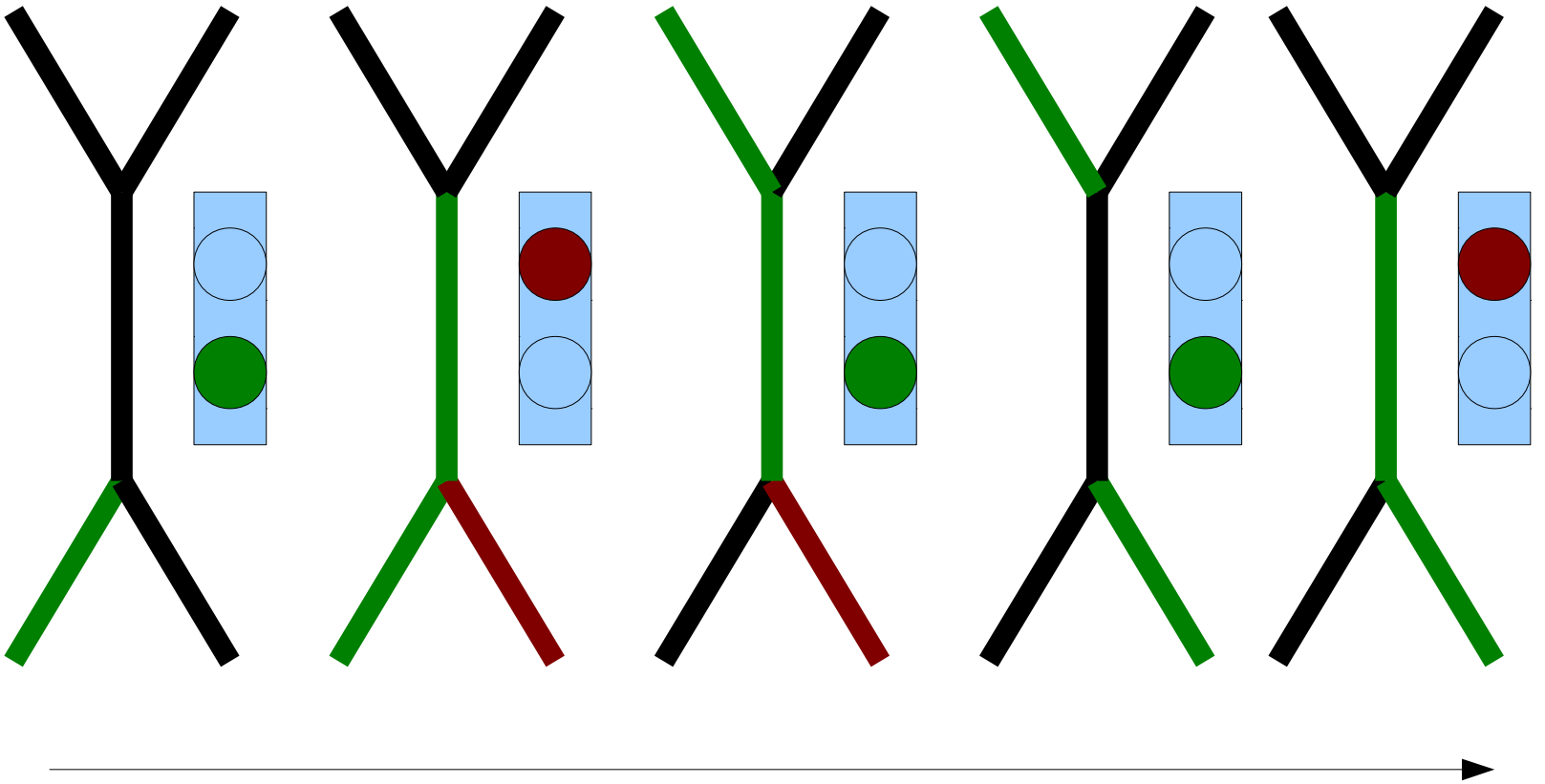
Semafor

- Zmienna całkowita ze zdefiniowanymi operacjami:
  - inicjacja liczbą dodatnią
  - **semWait** - operacja zmniejszenia wartości, wartość ujemna blokuje operację
  - **semSignal** - operacja zwiększenia wartości, jeżeli jest mniejsza lub równa 0, czekający proces jest odblokowany

Mutex

- Semafor binarny (mutex)
  - można go inicjalizować tylko na 0 lub 1
  - **semWait** – testuje semafor, wartość 0 blokuje proces, wartość jeden zmieniana jest na zero
  - **semSignal** – odblokowuje zablokowany proces lub ustawia wartość na 1

Mutex c.d.



Monitor

- Programowe rozszerzenie funkcjonalności semaforów
- Moduł składający się z procedur i danych lokalnych, oraz zapewniający że:
  - dostęp do danych lokalnych mają tylko procedury monitora
  - proces wchodzi do monitora wywołując jedną z jego procedur
  - tylko jeden proces może być wykonywany w monitorze, inne są zablokowane

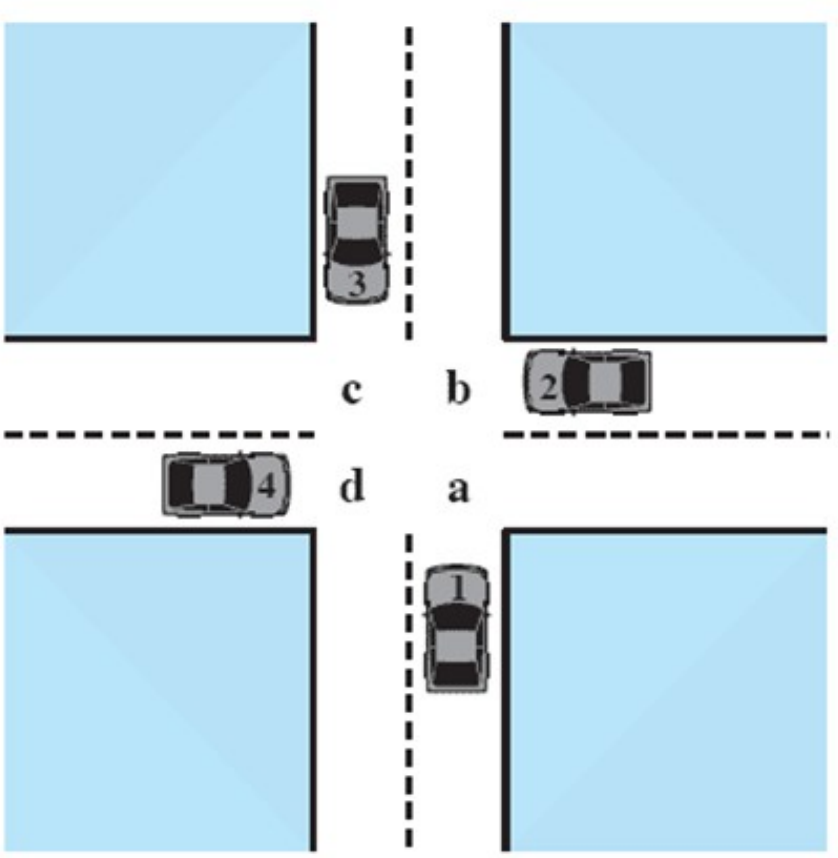
Komunikaty

- Przetwarzanie rozproszone – procesy dysponują osobnymi środowiskami wykonawczymi
- Komunikacja i synchronizacja procesów – poprzez wymianę komunikatów
- Przekazywanie komunikatów można też implementować w innych modelach przetwarzania.

Komunikaty c.d.

- Typowa implementacja: send/receive
- Synchronizacja:
  - synchroniczna wymiana komunikatów (blokująca)
  - asynchroniczna wymiana komunikatów (nieblokująca)

Impas



Stallings, „Systemy operacyjne ...”

Warunki wystąpienia impasu

- Wzajemne wykluczanie
  - Tylko jeden proces ma w danej chwili dostęp do zasobu.
- Wstrzymanie i oczekiwanie
  - Proces może trzymać rezerwację zasobu czekając na inne.
- Brak wywłaszczania
  - Nie ma możliwości odebrania procesowi zasobu
- Cykliczne oczekiwanie
  - Istnieje łańcuch procesów taki, że każdy z nich przetrzymuje zasób potrzebny innemu procesowi

Zapobieganie impasom

- Rezygnacja z wzajemnego wykluczania
  - Generalnie niepraktyczne (problem rywalizacji o zasoby).
  - Możliwość częściowej rezygnacji (np. równoległy odczyt z pliku przez kilka procesów).
- Zapobieganie przetrzymywaniu zasobów
  - Proces musi rezerwować wszystkie zasoby jednocześnie. Brak jednego zasobu powoduje zwolnienie pozostałych.
  - Problemy wydajnościowe
  - Problemy z implementacją (informacja o potrzebnych zasobach nie zawsze jest dostępna z góry).

Zapobieganie impasom

- Wywłaszczanie
  - SO może odebrać zasób danemu procesowi i przydzielić innemu.
  - Praktyczne tylko jeżeli stan zasobu można łatwo zachować i odtworzyć.
- Cykliczne oczekiwanie
  - Wymuszenie kolejności rezerwacji zasobów.
  - Problemy z wydajnością

Unikanie impasu

- Zapobieganie impasom to podejście statyczne - „Wbudowane” w projekt SO i obniżające wydajność.
- Unikanie impasu – działanie dynamiczne.
  - Decyzja o przydziale zasobu na podstawie aktualnej sytuacji.
  - Zasób nie zostanie przydzielony jeżeli może to doprowadzić do impasu.

Wykrywanie impasu

- Zapobieganie i unikanie impasu to podejścia konserwatywne. Mogą prowadzić do problemów z implementacją i wydajnością.
- Inne podejście – brak ograniczeń w przydziale zasobów i wykrywanie ewentualnego impasu.
- W przypadku zaistnienia impasu realizowana jest procedura usuwania impasu.

Algorytmy usuwania impasu

- Przerwanie wszystkich procesów w impasie.
- „Wycofanie” do ostatniego zapisanego stanu (rollback).
- Przerywanie procesów w impasie kolejno aż do usunięcia impasu.
- Wywłaszczanie procesów kolejno aż do usunięcia impasu.

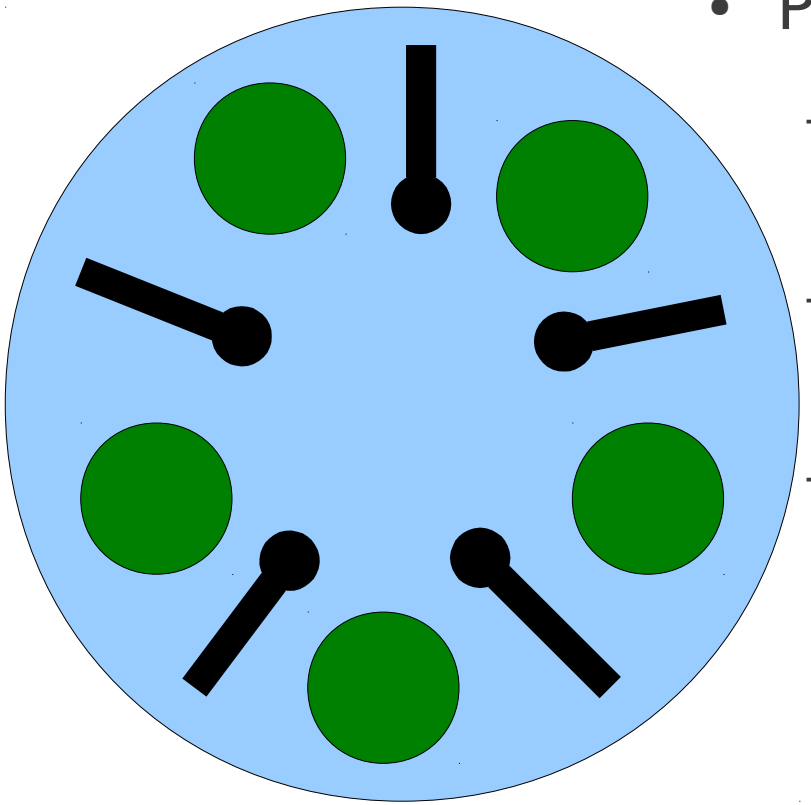
Klasyczne problemy synchronizacji

- Problem producent/konsument
  - Dwa typy procesów: producent – generuje dane, konsument – zużywa dane
  - Dane umieszczane są i pobierane z bufora
  - Problem: synchronizacja zadań w celu uniknięcia przepełnienia bufora lub próby pobrania danych z pustego bufora.

Klasyczne problemy synchronizacji c.d.

- Problem czytelników i pisarzy
  - Dwa typy procesów: czytelnicy (dostęp niemodyfikujący do zasobu) i pisarze (dostęp modyfikujący)
  - Czytelnicy mają nieograniczony dostęp, pisarze wymagają zasobu na wyłączność
  - Różne warianty: uprzywilejowani czytelnicy lub pisarze.
  - Problem: unikanie zagłodzenia czytelnika lub pisarza

Klasyczne problemy synchronizacji c.d.

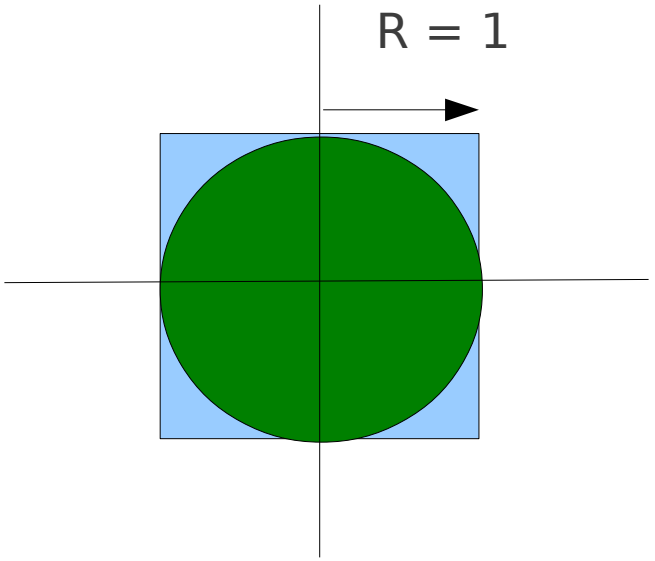


- Problem ucztujących filozofów
  - Okrągły stół i naprzemiennie leżące talerze i widelce
  - Filozofowie siedzą przy stole i oddają się medytacji.
  - Co jakiś czas filozof posila się korzystając z talerza i dwóch widelców (dobieranych pojedynczo)
  - Występują problemy związane z rywalizacją o zasoby, impasem, zagłodzeniem

PRACA DOMOWA

- Napisać w C/C++ program wyliczający liczbę PI metodą Monte Carlo (algorytm na następnym slajdzie)
- Program powinien być zoptymalizowany pod kątem szybkości
- Program powinien pozwalać na łatwą zmianę liczby próbkowań.

Algorytm



¼ Pkw = R*R

¼ Pkola = ¼ * PI * R * R

Pkola/Pkw = ¼ * PI

PI = 4 * Pkola/Pkw