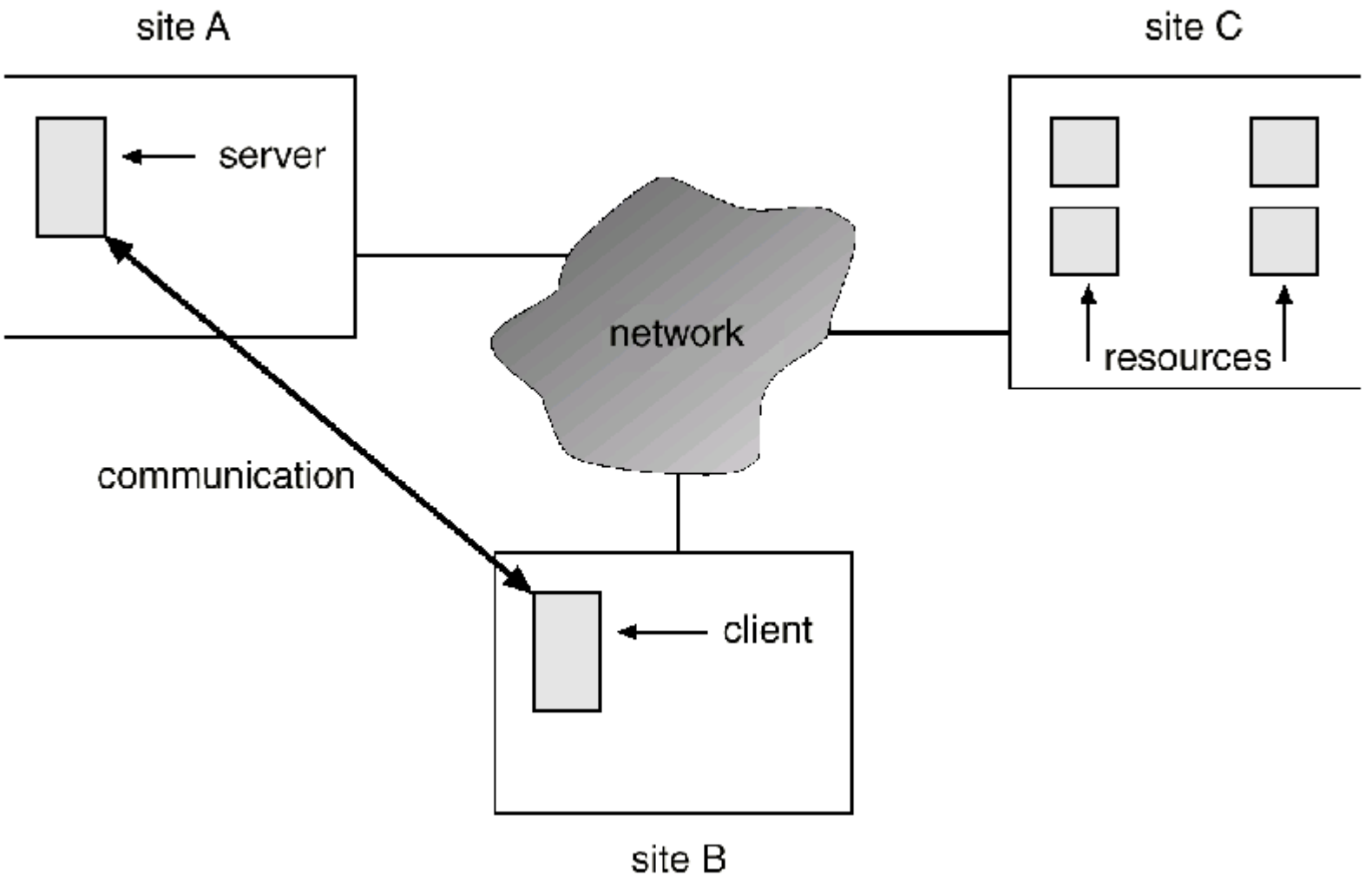
Wykład 14

Systemy rozproszone, Systemy czasu-rzeczywistego i multimedialne

System rozproszony (ang. distributed)



System składający się z wielu maszyn połączonych siecią

Systemy rozproszone - motywacja

Współdzielenie zasobów. Przykłady:

- Współdzielenie jednej drukarki, podłączonej do jednego komputera.
- Przetwarzanie informacji przez serwer bazy danych.
- Wykorzystanie specyficznych urządzeń sprzętowych

Przyspieszenie obliczeń – współdzielenie obciążenia (ang. *load sharing*)

- System obciążony obliczeniami może przekazać część swoich obliczeń innym systemom.

Zwiększenie niezawodności. Inne maszyny mogą przejąć cześć funkcji maszyny uszkodzonej.

- Wykrycie awarii.
- Przekazanie funkcji uszkodzonej maszyny innym.
- Reintegracja uszkodzonej maszyny po naprawie

Komunikacja – przesyłanie wiadomości pozwala na tworzenie nowych usług (np. poczta elektroniczna)

Sieciowe systemy operacyjne

Jest to powszechnie obecnie stosowane podejście do systemów operacyjnych dla komputerów pracujących w środowisku rozproszonym. Polega ono na uzupełnieniu istniejącego systemu (np. Unix, Windows) o funkcje związane z obsługą sieci.

- Zdalne logowanie i praca na innej maszynie.
- Transfer plików do/i zdalnej maszyny.
- Sieciowy system plików (zaimplementowany na jednej maszyn)
- e-mail, www, etc.

Generalnie w tym podejściu użytkownicy są świadomi istnienie wielu maszyn, a korzystanie z zasobów innej maszyny odbywa się w sposób jawny.

Systemy rozproszone

W tego typu systemach korzystanie z zasobów zdalnych wygląda tak samo, jak korzystanie z zasobów lokalnych.

Użytkownicy nie są świadomi istnienia wielu maszyn.

Migracja procesów. Proces (lub jego część) z maszyny obciążonej może migrować do maszyn mniej obciążonych.

Migracja danych. Dane z maszyny zdalnej mogą migrować na maszyne lokalną.

Systemy rozproszone są obecnie przedmiotem intensywnych badań podstawowych i stosowanych (także na Wydziale Informatyki PB).

Przykład: OpenMosix cluster

Przykład: klastry wykorzystujące systemy kolejkowe.

Problemy koordynacji (synchronizacji) rozproszonej

Synchronizacja procesów w systemie rozproszonym jest znacznie trudniejsza niż w systemie zcentralizowanym.

System rozproszony nie jest wyposażony w centralny zegar i współdzieloną pamięć, w niektórych sytuacjach nie jest możliwe stwierdzenie które ze zdarzeń zaszło pierwsze.

W koordynacji rozproszonej wykorzystuje się mechanizmy przesyłania komunikatów. Wiemy że komunikat może być odebrany, dopiero po tym jak został wysłany. Dzięki temu możemy określić porządek zdarzeń.

Wiele metod opiera się na wykorzystaniu centralnego ***procesu koordynatora***. Np. w ten sposób można zaimplementować wzajemne wykluczanie. Proces chcący wejść do sekcji krytycznej wysyła komunikat ***z żądaniem*** do koordynatora. Jeżeli w sekcji krytycznej nie ma innego procesu koordynator odsyła ***potwierdzenie*** (wejścia do sekcji). Proces wychodzący z sekcji krytycznej wysyła do koordynatora ***zwolnienie*** (ang. release) Koordynator po odebraniu komunikatu zwolnienia, wysyła potwierdzenie do kolejnego procesu oczekującego do wejścia do sekcji.

Wykorzystanie centralnego koordynatora zmniejsza wydajność => typowe wąskie gardło.

Problem ***elekcji koordynatora*** – w systemie rozproszonym może dojść do awarii maszyny na której wykonuje się koordynator.

Algorytm tyrana

Zakładamy, że procesy są uporządkowane pod względem priorytetów: P_1, P_2, P_3, P_4, ...

Niech proces P_i po wysłaniu komunikatu do koordynatora nie doczeka się odpowiedzi przed upływem czasu T. P_i stwierdza, że koordynator uległ awarii i stara się sam wybrać na koordynatora.

P_i wysyła do wszystkich procesów o wyższym priorytecie komunikat informujący o elekcji, po czym czeka przez czas T na odpowiedź. Jeżeli nie ma odpowiedzi, to P_i zakłada, że procesy o wyższym priorytecie uległy awarii i został wybrany na koordynatora. W związku z tym wysyła informację o swojej elekcji procesom i niższych priorytetach.

Jeżeli jednak P_i otrzyma odpowiedź, to czeka przez czasu T' na informację że proces o wyższym priorytecie został koordynatorem. Jeżeli jej nie otrzyma, to rozpoczyna algorytm od początku.

Jeżeli P_i nie jest koordynatorem, to w każdej chwili może otrzymać jeden z dwóch komunikatów:

P_j został koordynatorm (j>i). P_i zapisuję te informację.

P_j usiłuje wybrać się na koordynatora (j<i). P_i wysyła odpowiedź do P_j, po czym sam rozpoczyna algorytm elekcji, o ile go jeszcze nie rozpoczął.

Klaster (ang. cluster)

Klaster to zbiór wielu niezależnych komputerów, połączonych siecią, sprawiających wrażenie jednej maszyny.

Klastry możemy podzielić na:

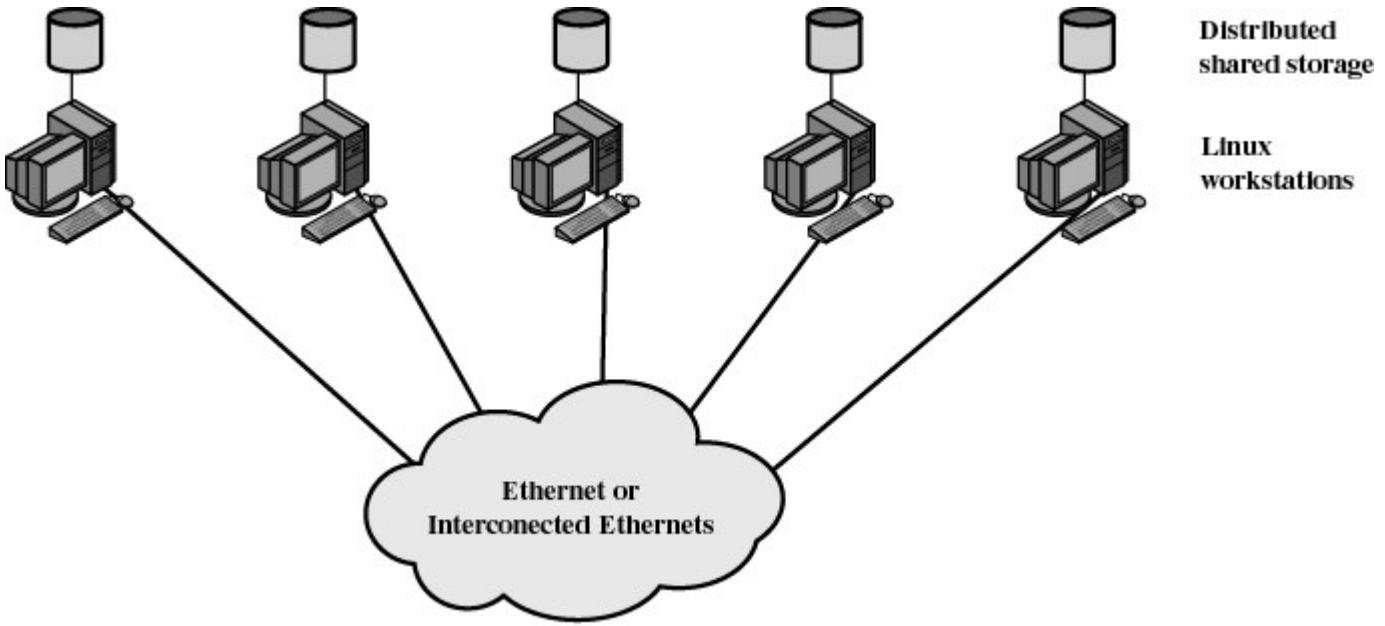
Służące celom obliczeniowym (ang. High Performance Computing) – wiele systemów wchodzących w skład klastra może rozwiązać szybko trudny problem obliczniowy.

Zapewniające redundancję (High Availability Computing) w przypadku awarii jednego z elementów klastra jego zadanie przejmowane są przez pozostałe.

Przykład: Klaster wielu serwerów obsługujących żądania HTTP. Kolejne żądania są dystrybuowane do poszczególnych serwerów. W przypadku awarii jednego z serwerów pozostałe przejmuję jego zadania (aczkolwiek wzrasta obciążenie).

Problem: co się stanie gdy przestanie działać router dystrubuujący żądanie HTTP. Potrzebujemy dwóch takich routerów (brak pojedynczego punktu awarii – single point of failure).

Klaster typu HPC



Wydzielona (o mocniejszej konfiguracji) maszyna pełni rolę węzła zarządzającego. Pozostałe są węzłami roboczymi.

Sieciowy system plików (np. NFS) – każdy użytkownik ma taką samą ścieżkę na dowolnym węźle (katalog /home)

Sieciowy system kont (np. NIS albo NIS+) każdy użytkownik ma na każdej maszynie taki sam uid, gid, hasło, etc..

Dzięki temu mogę się zalogować na dowolną maszynę klastra i pracować w identycznym środowisku (nie wiedząc nawet na jakiej maszynie)

Praca z klasterm typu HPC

Użytkownik loguje się (np. ssh) na węzeł zarządzający. Edytuje i kompiluje swój program, przygotowuje dane.

Następnie zleca zadanie do systemu kolejkowego, podając liczbę potrzebnych procesorów.

System kolejkowy (np. OpenPBS, Sun Grid Engine) czeka aż procesory stają się wolne, po czym przydziela węzły robocze i uruchamia na nich program.

Jeżeli wszystkie węzły są zajęte, zadanie musi czekać w kolejce aż jakieś się zwolnią.

Mamy tutaj do czynienia z systemem wsadowym (ang. batch), a system kolejkowy jest podręcznikowym przykładem planisty długoterminowego.

System ten wykonuje pewien algorytm szeregowania (np. Najpierw zadania potrzebujące najmniej procesorów, albo dwie kolejki jedna dla studentów a druga dla profesorów, etc ...)

Projekt Clusterix

Budowa super-klastra obliczeniowego dla polskiego środowiska naukowego

Finansowanie: 50% uczelnie + 50% Ministerstwo Nauki

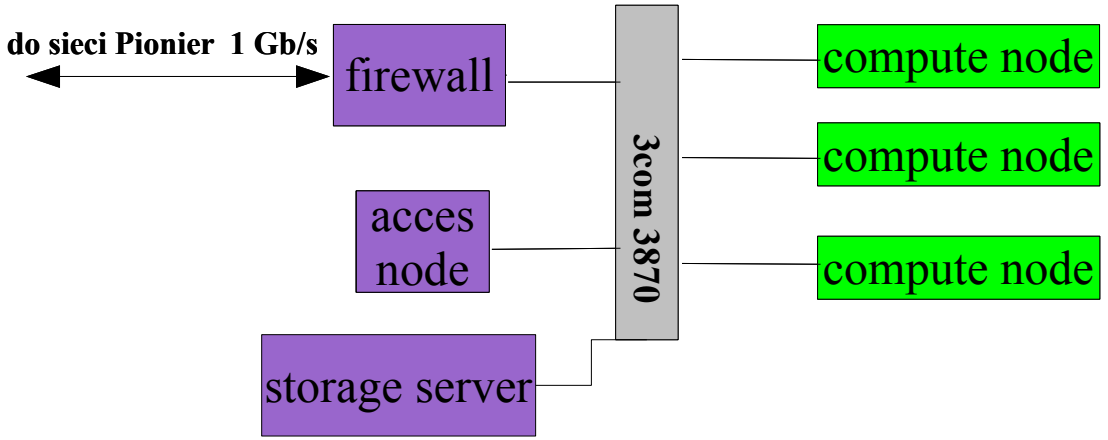
Realizowany przez 12 czołowych ośrodków akademickich w Polsce.

Białystok jest członkiem tego ekskluzywnego klubu !!!

12 klastrów połączonych sięcią optyczną PIONIER utworzy jeden olbrzymi komputer równoległy.

Wykorzystuje procesory 64-bitowe (Intel Itanium 2) oraz protokół IP v 6

Przykład - Klaster zbudowany w ramach projektu Clusterix www.clusterix.pl



Firewall i węzeł dostępowy: 2xPentium Xeon 2.4 GHz

Węzeł obliczeniowy: 2xItaniumII 1.4GHz

Storage server Pentium IV macierz Raid 5 8x250 GB

Połączenie wewnątrz klastra: dwa kanały GigabitEthernet na każdą maszynę - na poziomie aplikacji na razie 30-50MB/s w przyszłości około 100MB/s

Nowy klaster zakupiony w ramach programu ZPOOR

Jeden węzeł zarządzający

16 węzłów 2xXeon 3.2 GHz

sięć Infiniband 4x umożliwiająca na poziomie aplikacji transfer około 1GB/s

w budowie i zarządzaniu czynny udział biorą (brali) Państwa starsi koledzy (z Koła Naukowego)

Systemy czasu rzeczywistego - definicja

System czasu rzeczywistego to takim system, w którym poprawność zależy nie tylko od poprawności wyników, ale również od dostarczenia tych wyników przed upływem nieprzekraczalnego terminu (ang. deadline). Przykłady:

Komputer sterujący rakietą musi podjąć obliczyć nowy kurs. Jeżeli wyniki zostaną obliczone zbyt późno rakieta się rozbije.

Odtwarzacz DVD musi zdekodować klatkę w ciągu 25 ms (50 klatek na sekundę). W przeciwnym wypadku nie da się utrzymać płynnej animacji.

Komputer sterujący pralką musi wyłączyć pompę możliwie szybko po otrzymaniu sygnału o napełnieniu zbiornika. Jeżeli zrobi to po 10 minutach to pralka zostanie zalana.

Powyższa definicja odnosi się do systemu typu hard real-time.

Większość systemów dla komputerów biurkowych i serwerów (np Uniks, Windows) to systemy typu soft real-time nie gwarantujące dotrzymania terminów. Zamiast tego zdefiniowano w nich specjalną klasę procesów: procesy klasy czasu rzeczywistego. Procesy tej klasy zawsze otrzymują procesor przed innymi procesami.

„Zwykły” system operacyjny jako systemu czasu rzeczywistego

Wykorzystanie typowego (Uniks, Windows) systemu operacyjnego to zadań typu hard real-time jest niezwykle trudne. Głównym powodem jest duży i niesprecyzowany ***czas reakcji na zdarzenie***. (event latency).

Czas ten definiujemy jako czas od momentu zajścia zdarzenia do momentu wygenerowania odpowiedzi na zdarzenie

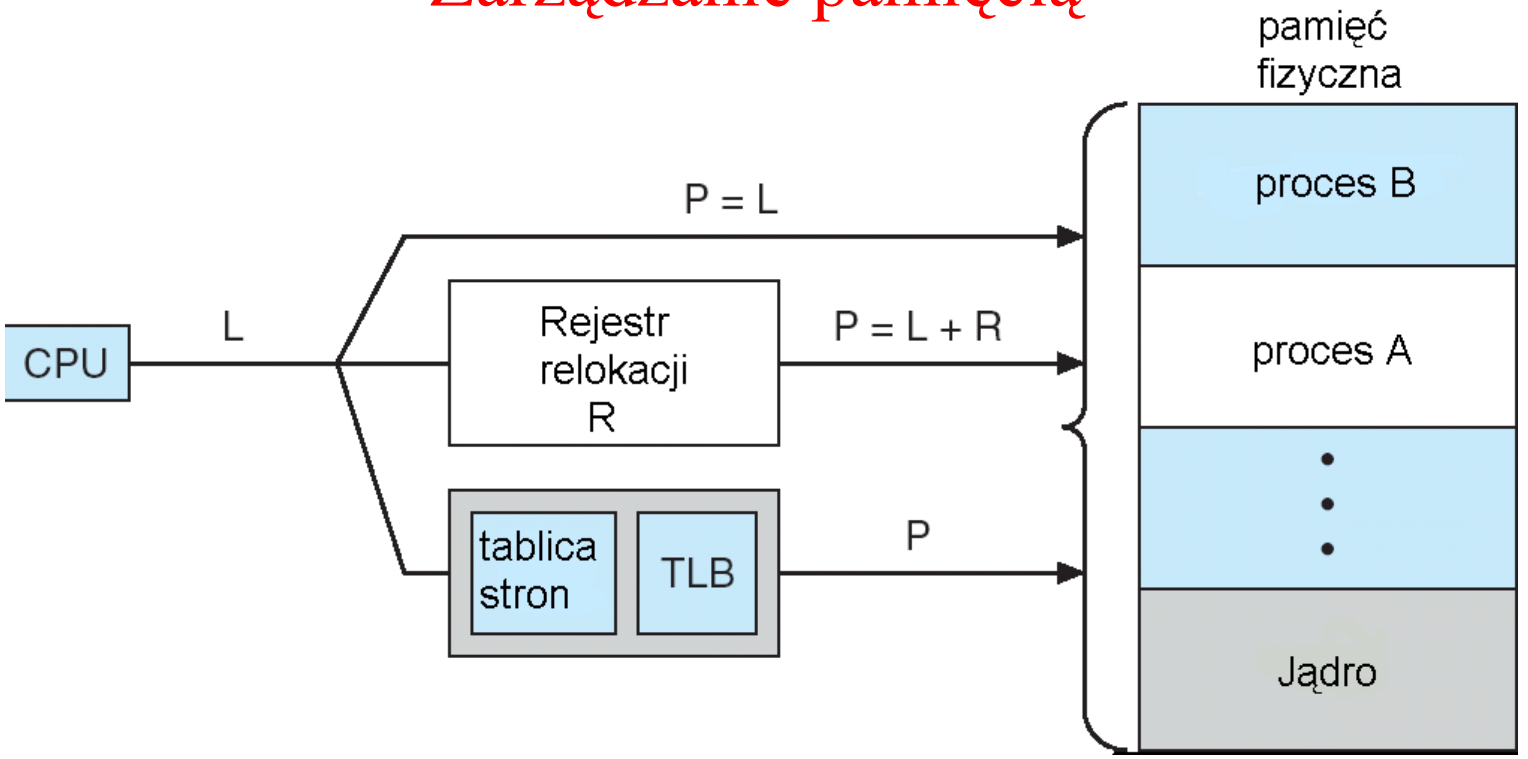
W zwykłych systemach czas ten jest opóźniany przez następujące czynniki:

Mechanizm pamięci wirtualnej sprawia, że część stron procesu obsługującego zdarzenie może być przechowywana na dysku (np. w pliku wymiany). Czas sprowadzenie tych stron jest horrendalnie duży (jak na systemy czasu rzeczywistego).

Wiele systemów wykorzystuje ***niewywłaszczalne jądro.*** Oznacza to, że proces wykonujący się w trybie jądra nie może zostać wywłaszczony. Kiedy zdarzenie nastąpi podczas wykonywania się takiego procesu, proces wysokopriorytetowy proces czasu rzeczywistego musi czekać aż aktualnie wykonujący się proces albo zrzeknie się procesora (uśpi się) albo powróci do trybu użytkownika.

Ponadto zwykłe systemy wyposażone są w wiele mechanizmów (np. graficzny interfejs użytkownika, pamięć wirtualna, systemy plików) niepotrzebnych w systemach czasu rzeczywistego i nie udostępnionych przez sprzęt wykorzystywany w systemach czasu rzeczywistego. Często hardware jest typu System on a Chip (procesor+pamięc+układy peryferyjne w jednym układzie scalonym)

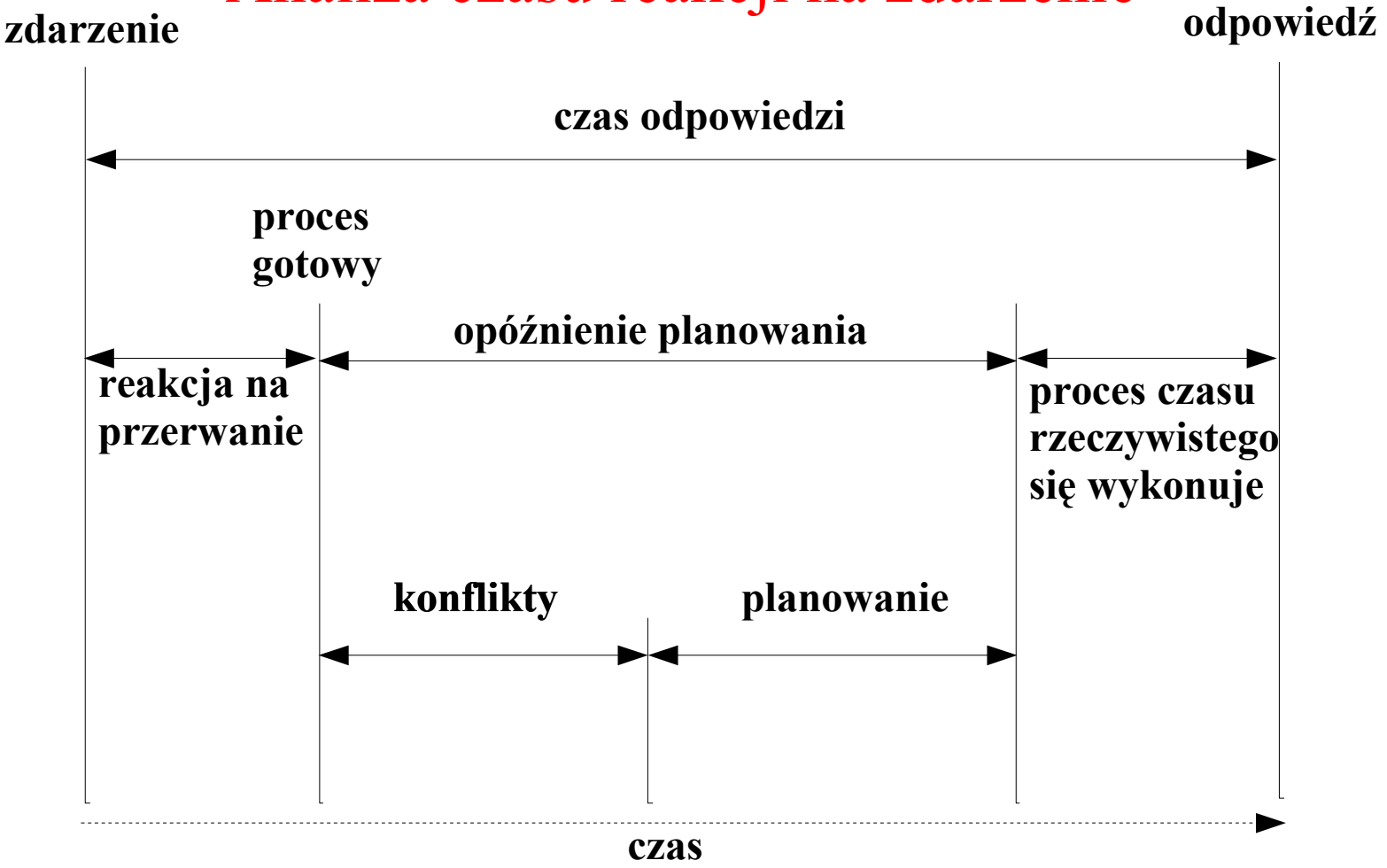
Zarządzanie pamięcią



Wsparcie sprzętowe do zarządzania może być (a) nieistniejące – adres logiczny staje się adresem fizycznym (b) opierać się na rejestrze relokacji (c) pełnym wsparciem dla stronicowania.

W przypadku (c) i prostszych systemów koszt sprzętowy jednostki zarządzającej pamięcią może być porównywalny np. z sumarycznym kosztem pozostałych elementów rocesora.

Analiza czasu reakcji na zdarzenie



Reakcja na przerwania – od momentu zgłoszenia przerwania do momentu wywołania handlera (blokowanie przerwań zwiększa ten czas).

Konflikty – wywłaszczenie procesu wykonującego się w trybie jądra (lub oczekiwanie na jego wyjście z jądra).

Opóźnienie planowania => jądro powinno być wywłaszczalne (np. w Solarisie 100 ms przy jądrze niewywłaszczalnym > 100ms, przy wywłaszczalnym <1ms)

Planowanie w systemach czasu rzeczywistego

Przyjmujemy periodyczny (okresowy) model procesów. Każdy proces może być opisany przez następujące parametry:

okres p tzn. czas pomiędzy kolejnymi zdarzeniami wymagającymi obsługi przez proces

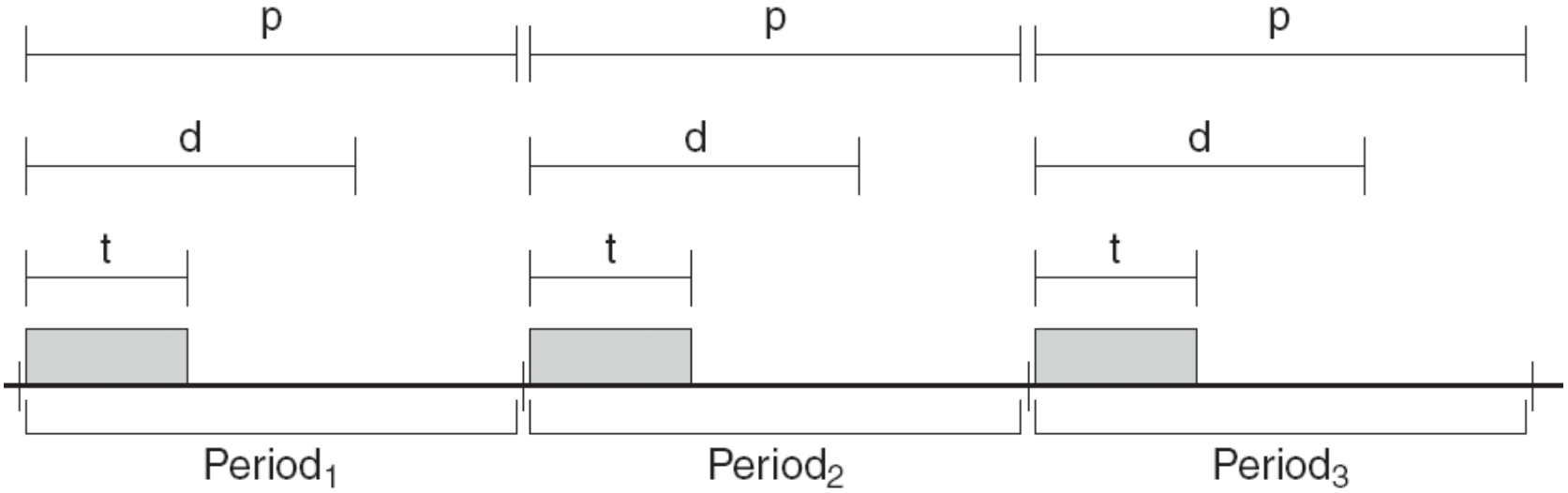
termin d w którym zdarzenie musi być obsłużone (od momentu zajścia zdarzenia)

czas t potrzebny procesowi na obsługę zdarzenia

Zachodzi relacja $0 \leq t \leq d \leq p$

Stopień wykorzystania procesora jest równy $u=t/p$. Warunek konieczny wykonywalności szeregowania: suma stopni wykorzystania procesora $\sum u \leq 1$.

Proces oznajmia swoje parametry t,d,p planiście. Planista albo podejmuje się wykonania procesu gwarantując dotrzymania terminu albo odrzuca proces.



Rate-monotonic scheduling

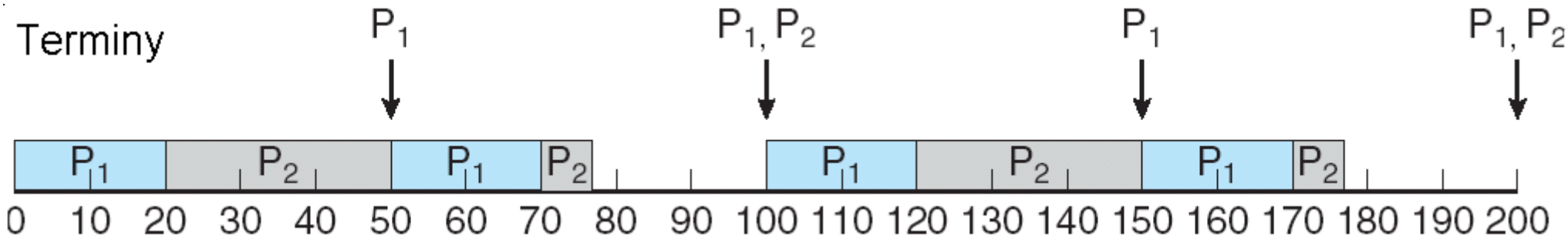
Procesy są planowane na podstawie statycznego priorytetu równego częstotliwości zdarzeń (1/p – odwrotność procesu). Proces o wyższym priorytecie wywłaszcza proces o niższym priorytecie.

Przykład: dwa procesy:

P1: p=50, d=50, t=20 (P1 ma krótszy okres => większą częstotliwość i priorytet)

P2: p=100, d=100, t=35

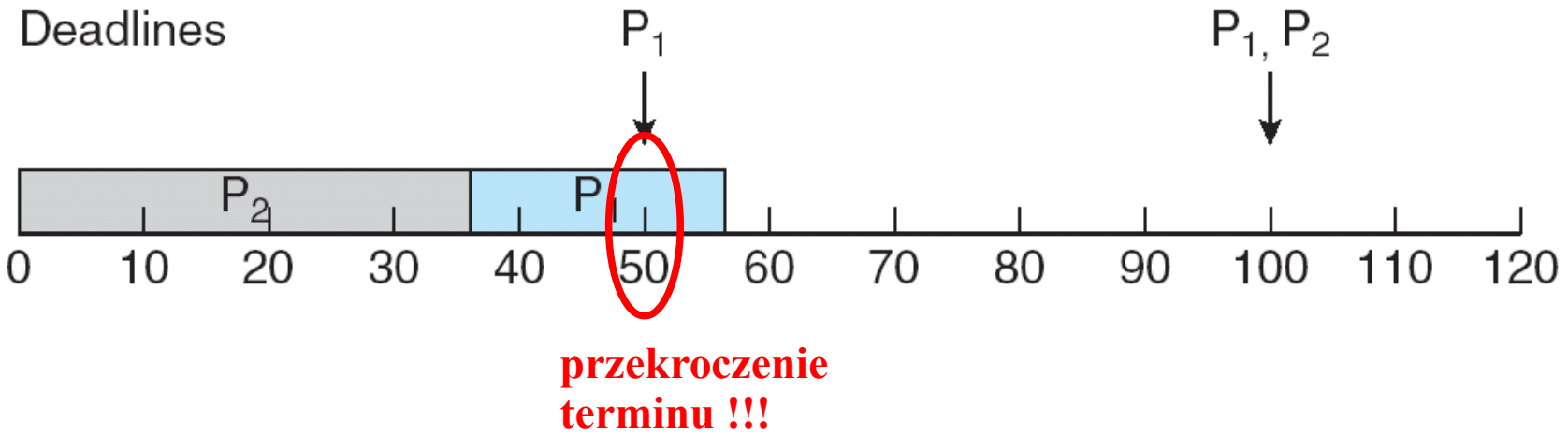
Całkowite obciążenie procesora (20/50) + (35/100)=0.75



Właściwości algorytmu (1)

Spośród klasy algorytmów z priorytetami statycznymi jest to algorytm optymalny, w takim sensie, że jeżeli nie dotrzymuje terminów, to żaden inny algorytm z tej klasy również nie dotrzyma terminów.

Przykład: zakładamy, że proces P2 ma większy priorytet:



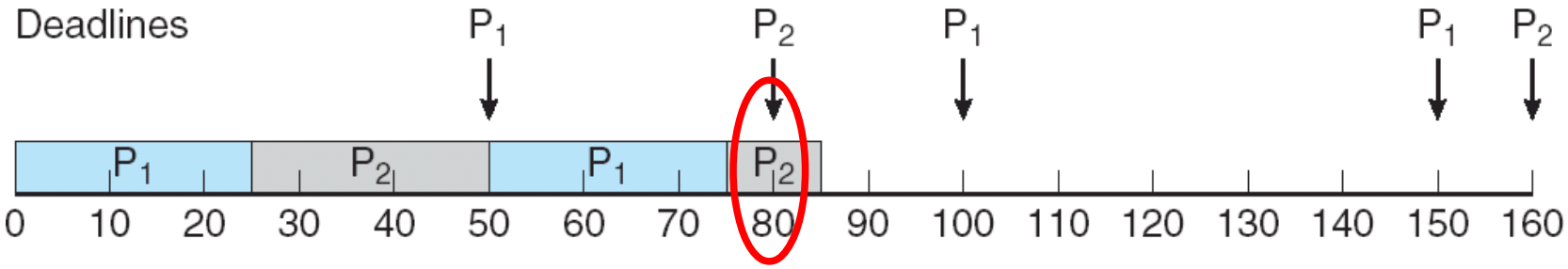
Właściwości algorytmu (2)

Przykład, w którym dotrzymanie terminów nie jest możliwe:

P1: p=50, d=50, t=25 (wyższy priorytet)

P2: p=80, d=80, t=35.

Całkowite wykorzystanie procesora (25/50)+(35/80)=0.94. Wydaje się że procesy powinniśmy być w stanie zaplanować.



przekroczenie terminu !!!

W pesymistycznym przypadku algorytm nie gwarantuje dotrzymania terminów, gdy całkowite wykorzystanie procesora:

$$u \geqslant 2\left(2^{1/N}-1\right)$$

Dla liczby procesów N=2 otrzymujemy u≈0.83

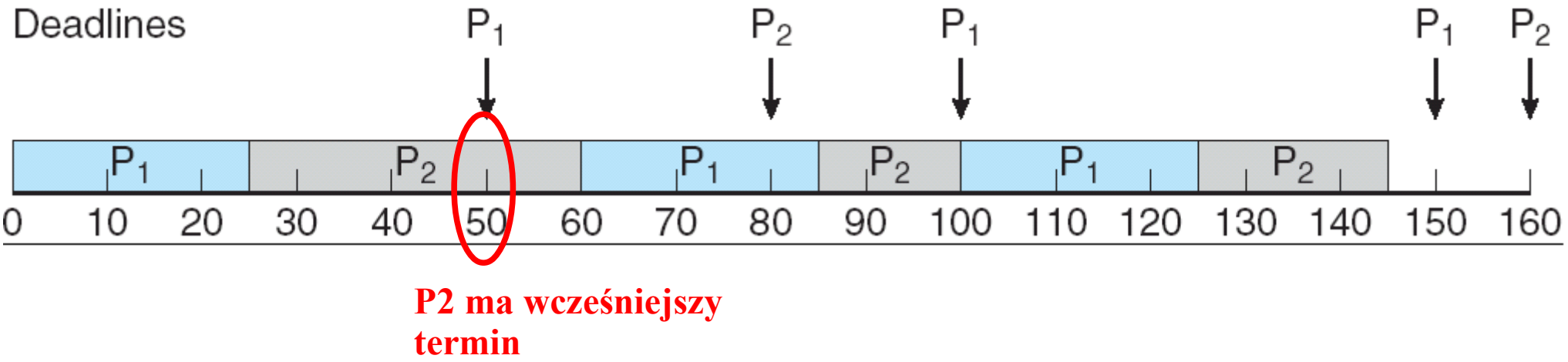
Algorytm najwcześniejszy termin najpierw (ang. earliest deadline first - EDF)

Priorytet przypisywany dynamicznie – w momencie

Przykład, dane podobne jak poprzednio:

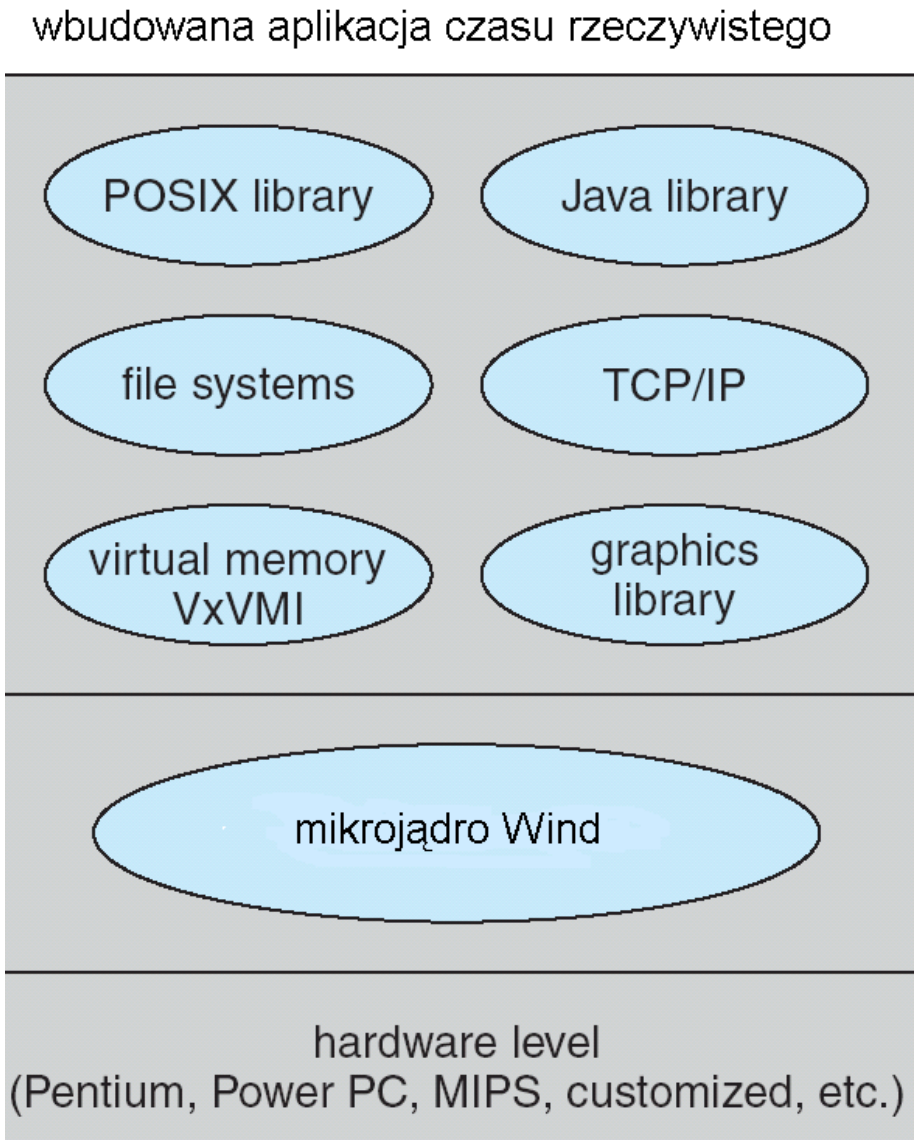
P1: p=50, d=50, t=25 (wyższy priorytet)

P2: p=80, d=80, t=35.



P2 ma wcześniejszy termin

Przykład systemu czasu-rzeczywistego: VxWorks 5.x



Opcjonalne podsystemy: grafiki, systemy plików, Java, sieci TCP/IP, biblioteka Posix, pamięć wirtualna. Pozwala to na minimalizację zajętości (ang. footprint) pamięci.

Mikrojądro (ang. microkernel) Wind

wywłaszczalne

gwarantowany czas reakcji na przerwania

Zastosowanie: samochody, urządzenia konsumenckie, przełączniki sieciowe oraz ***Marsjańskie łaziki Spirit i Opportunity***.

Procesy czasu – rzeczywistego w systemach Uniksowych

Standard POSIX1.b określa dwie klasy procesów czasu rzeczywistego: SCHED_FIFO oraz SCHED_RR.. Dodatkowo każdy proces ma statyczny priorytet z zakresu 1-99.

Procesy obydwu klas czasu rzeczywistego mają bezwzględny priorytet nad zwykłymi procesami (klasa SCHED_OTHER).

Procesy klasy SCHED_FIFO planowane są na podstawie algorytmu FCFS. Procesowi klasy SCHED_FIFO o najwyższym aktualnie priorytecie statycznym w kolejce procesów gotowych nie zostanie odebrany procesor.

Procesy klasy SCHED_RR (RR to round robin) o identycznym priorytecie statycznym są planowane w ramach algorytmu rotacyjnego.

System Uniksowy zgodny ze standardem POSIX1.b jest systemem klasy soft real-time.

Systemy multimedialne

Dane multimedialne obejmują klipy audio oraz wideo (np. mp3 i mpeg) oraz transmisje sieciowe na żywo (ang. webcasts).

Dane multimedialne są przechowywane w systemie plików tak jak zwykłe dane. Jednakże dostęp do nich musi być zagwarantowany z uwzględnieniem terminów. W typowej transmisji klient-serwer obejmuje to:

- dostęp i odczyt danych z dysku.
- przesłanie danych przez sieć.
- dekompresję.

Pliki multimedialne różnią się znacznie od „zwykłych” plików w systemie.

- znaczny rozmiar
- sekwencyjny dostęp.

Typowe aplikacja: video-on-demand. Serwer musi dostarczyć wielu klientom indywidualne strumienie danych multimedialnych.

Klient żąda spełnienia zbióru wymagań nazywanego jakością obsługi ang. (Quality of Service QoS).

QoS w systemach multimedialnych

W systemie multimedialnym QoS jest określany przy pomocy następujących parametrów: (a) przepustowość (np. liczba klastek na sekundę dostarczonych klientowi) (b) wiarygodność (np. sposób obsługi błędów w sieci) (c) wariancja (np. odnośnie przepustowości).

Zapewnienie odpowiedniego QoS wymaga podjęcia akcji na poziomie: (a) planowania procesora (b) systemu plików i planowania dysków (c) protokołów sieciowych

QoS może być zapewnione na trzech poziomach:

Best-effort QoS system podejmuje wszelkich możliwych starań, ale nie ma gwarancji spełnienia wymagań.

Soft QoS system priorytezuje zadania, nadając pewnym większy priorytet, ale dalej nie ma gwarancji spełnienia wymagań.

Hard QoS gwarantuje spełnienie wymagań.

QoS może być negocjowany pomiędzy serwerem a klientem. Ponadto serwer na ogół jest wyposażony w kontrolę przyjęcia (ang. admission control), która odrzuca żądanie klienta gdy serwer nie jest w stanie zagwarantować Hard QoS.

Planowanie dysku w systemach multimedialnych

Większość systemów używa planowania opartego na algorytmach SCAN albo C-SCAN. Takie planowanie jest nieczułe na terminy i nie nadaje się do zastosowań multimedialnych.

Z kolei bezpośrednie zastosowanie algorytmu EDF prowadziłoby do kompletnego braku jakiejkolwiek minimalizacji czasu przemieszczania głowicy dysku (ang. seek time).

Stosuje się kombinacje powyższych technik np. algorytm SCAN-EDF. Algorytm sortuje żądania w porządku EDF, a żądania o takich samych terminach obsługuje w porządku SCAN.

Jak należy postąpić w przypadku, gdy mamy kilka żądań o niewiele różniących się terminach położonych relatywnie blisko siebie. Jednym z możliwych rozwiązań jest grupowanie (ang. batch) żądań.

Np. podział czasu na kwanty o stałej długości, żądania których termin zalicza się do kwantu są łączone w grupy i obsługiwane np. za pomocą algorytmu SCAN.

SCAN-EDF -przykład

żądanie	termin	cylinder
A	150	25
B	201	112
C	399	95
D	94	31
E	295	185
F	78	85
G	165	150
H	125	101
I	300	85
J	210	90

Głowica jest nad cylindrem 50, porusza się w stronę cylindra 51, długość kwantu 100 ms.

Grupujemy żądania w 4 grupy: 0-99ms (D,F); 100-199ms (A,G,H); 200-299ms (B,E,J); 300-399ms (C,I).

Wewnątrz każdej grupy obsługa według algorytmu SCAN.

Finalny porządek:

(F, zmiana kierunku,D)

(A, zmiana kierunku, H, G)

(E, zmiana kierunku B,J)

(I, zmiana kierunku C)