Budowa i oprogramowanie komputerowych systemów sterowania

Wykład 9

Systemy operacyjne

Pamięć wirtualna

- Pamięć wirtualna jest techniką programową a także sprzętową gospodarowania pamięcią operacyjną RAM pozwalającą na przydzielanie pamięci dla wielu procesów, zwalnianie jej i powtórne przydzielanie.
- Mechanizm umożliwia przydzielenie procesom więcej pamięci niż rzeczywista ilość pamięci fizycznej zainstalowanej w komputerze poprzez przeniesienie danych z ostatnio nie używanej pamięci do pamięci masowej (np. twardego dysku).
- W sytuacji gdy procesor odwołuje się do danych z pamięci przeniesionej na dysk przesuwa się te dane do pamięci w wolne miejsce, a gdy brak wolnej pamięci zwalnia się ją przez wyżej opisane przerzucenie jej na dysk.

Pamięć wirtualna

- Najczęściej spotykane są dwa sposoby przechowywania danych zrzuconych z pamięci fizycznej na dysk.
- Pierwszy, stosowany w systemach rodziny Windows polega na zapisie pamięci w pliku (znajdującym się na ustalonej partycji komputera).
- Drugi, stosowany w systemach z rodziny UNIX to utworzenie osobnej partycji wymiany (partycji swap) przeznaczonej wyłącznie na pamięć wirtualną.
- Zapewnia to szybszy dostęp do danych niż pierwsze rozwiązanie (głównie ze względu na ominięcie obsługi systemu plików).

System operacyjny

- System operacyjny jest głównym programem integrującym sprzętowe i programowe zasoby komputera, który działa jako pośrednik między użytkownikiem komputera a sprzętem komputerowym.
- Zadaniem systemu operacyjnego jest tworzenie środowiska, w którym użytkownik może przygotowywać i wykonywać inne programy.
- Podstawowym celem systemu operacyjnego jest:
  - zarządzanie zasobami sprzętowymi komputera
  - zapewnienie wygodnej eksploatacji systemy komputerowego
  - wydajna eksploatacja sprzętu komputerowego

Wady ogólnych systemów operacyjnych

- W systemach operacyjnych ogólnego przeznaczenia dąży się do optymalizacji średniej wydajności, kosztem mniejszej efektywności w przypadku pojedynczego zadania.
- Udogodnienia takie jak stronicowanie na żądanie i wszelkie pamięci podręczne powodują nieprzewidywalne odchylenia czasu wykonania.
- Szeregowanie zadań nie uwzględnia wymagań odnośnie czasu wykonania określonego zadania.

System czasu rzeczywistego

- System czasu rzeczywistego (ang. real-time) jest systemem, który jest stosowany tam gdzie występują surowe wymagania na czas wykonywania operacji lub przepływu danych.

- Podział systemów czasu rzeczywistego:
  - łagodny system czasu rzeczywistego (ang. soft real-time system)
  - rygorystyczny system czasu rzeczywistego (ang. hard real-time system)

Proces

- Proces – jest to wykonywany program, który ma przydzieloną pamięć, licznik rozkazów, zestaw rejestrów, zasoby wejść-wyjść.
- W systemie może istnieć kilka procesów związanych z wykonywaniem tego samego programu.
- Stany procesu:
  - nowy
  - aktywny
  - oczekiwanie
  - gotowy
  - zakończony

Proces

- Planowanie procesów – uruchomione w systemie procesy trafiają do kolejki zadań (ang. job queue), gotowe do działania procesy trzymane są na liście zwanej listą procesów gotowych (ang. ready queue), w danej chwili może być aktywny tylko jeden proces.

- Przełączanie kontekstu (ang. context switch) – czynności związane z przełączaniem procesora pomiędzy wykonywaniem kolejnych procesów, jest to czas wykorzystywany przez system operacyjny na zapamiętanie stanu starego procesu i odtworzenie stanu nowego procesu, w zależności od systemu wynosi od 1 do 100us.

Wątek

- Wątek (ang. thread) – podstawowa jednostka wykorzystania procesora posiadająca licznik rozkazów, zbiór rejestrów i obszar stosu. Wątek współużytkuje z innymi równorzędnymi wątkami sekcję kodu, sekcję danych oraz zasoby systemu operacyjnego, co określane jest zadaniem (ang. task).

Zadania

- Wywłaszczanie – możliwość zatrzymania aktualnego zadania, zmiany kontekstu procesora i uruchomienie innego zadania.
- Priorytet zadania – liczba określająca względną pilność zadania
- Praca wielozadaniowa – możliwość obsługi przez system operacyjny więcej niż jednego zadania umieszczonego w pamięci

Szeregowanie zadań

- Szeregowanie zadań – planowanie przydziału procesora dla procesów czekających w kolejce procesów gotowych

- Algorytmy szeregowania zadań:
  - FCFS pierwszy zgłoszony pierwszy obsłużony
  - SJF najpierw najkrótsze zadanie
  - planowanie priorytetowe
  - planowanie rotacyjne (ang. round-robin) - zaprojektowane specjalnie dla systemów z podziałem czasu, każdemu zadaniu w kolejce przydzielany jest kwant czasu, zwykle 10-100ms, po upływie tego czasu proces jest wywłaszczany

Semafory

- Semafor jest zmienna całkowita, której wartość może być ustawiana przez jeden proces, a sprawdzana i kasowana przez drugi.
- Ustawienie zmiennej jest traktowane przez proces sprawdzający jako sygnał spełnienia warunku umożliwiającego dalsze wykonanie tego procesu.
- Brak spełnienia warunku w chwili sprawdzenia oznacza niemożliwość kontynuacji procesu i konieczność zawieszenia jego wykonania.
- Późniejsze ustawienie wartości zmiennej powoduje automatyczne wznowienie zawieszonego procesu.

Semafory

- Dokładna definicja semafora wprowadzona w standardzie POSIX precyzuje, że warunkiem umożliwiającym kontynuacje procesu sprawdzającego jest dodatnia wartość semafora, natomiast wartość mniejsza lub równa zeru powoduje zawieszenie tego procesu.
- Dla uniknięcia potencjalnych błędów, wynikających z jednoczesnego działania na tej samej zmiennej, operacje sprawdzające i ustawiające wartość semafora muszą być realizowane przez niepodzielne funkcje systemowe.

Depozyty

- Depozyty można uważać za rozszerzenie tradycyjnego mechanizmu semaforów służących do przekazywania impulsów synchronizujących między współpracującymi procesami.
- Rozszerzenie operacji polega na stowarzyszeniu impulsów z pewna stała i niezmienna wiadomością, przekazywana do odbiorcy niezależnie od normalnej akcji synchronizującej.

Potoki i kolejki

- Funkcjonalnie, obydwa mechanizmy realizują niemal identyczne bufory komunikacyjne, umożliwiające magazynowanie i jednokierunkowe przekazywanie danych miedzy procesami.
- W obydwu wypadkach dane są zapisywane do bufora i odczytywane w tej samej kolejności, w postaci strumienia przekazywanych bajtów.
- Operacja zapisania danych nie wstrzymuje nadawcy, natomiast operacja odczytu z pustego bufora może spowodować zawieszenie odbiorcy do czasu pojawienia się danych.

Potoki i kolejki

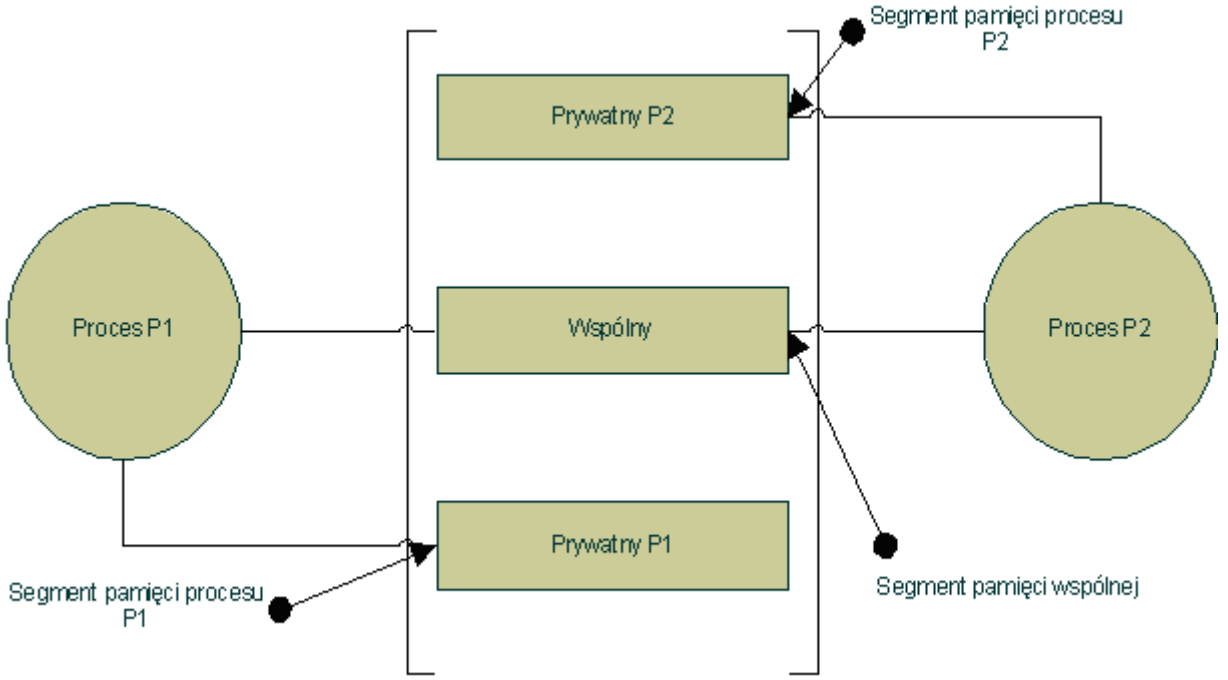
- Mimo funkcjonalnego podobieństwa, obydwa mechanizmy różnią się istotnie na poziomie implementacji.
- Kolejki są realizowane jako specjalne pliki dyskowe z buforowanymi operacjami zapisu i odczytu.
- Pliki te istnieją permanentnie na dysku, a ich nazwy sa zapisane w katalogu.
- Wszystkie operacje na kolejkach wykonuje systemowy menedżer plików.
- Implementacja potoków może być dwojaka, zależnie od tego, czy będą one organizowane przez menedżer plików, czy specjalny menedżer potoków.
- W pierwszym wypadku, potoki są realizowane jako tymczasowe pliki dyskowe bez nazwy, z buforowanymi operacjami zapisu i odczytu. W drugim, dane są przekazywane wyłącznie poprzez bufory w pamięci, bez zapisywania ich na dysku.

Sygnały

- Sygnały wykorzystywane są najczęściej do informowania o błędach i pewnych sytuacjach wyjątkowych, jak np. wznowienie zawieszonego procesu lub pilne zdarzenie na łączu TCP/IP. Niektóre z sygnałów powodują awaryjne zakończenie procesu.
- Obsługa sygnału jest zazwyczaj asynchroniczna - w momencie jego nadejścia wywoływana jest odpowiednia procedura obsługi, o ile taka istnieje dla danego typu sygnału

Pamięć współdzielona

- Procesy mają rozdzielone segmenty danych, modyfikacje na danych w jednym procesie nie mogą wpłynąć w żaden sposób na dane drugiego procesu. Aby oba procesy miały dostęp do tych samych danych należy stworzyć oddzielny segment danych i nadać dostęp do niego obu procesom.

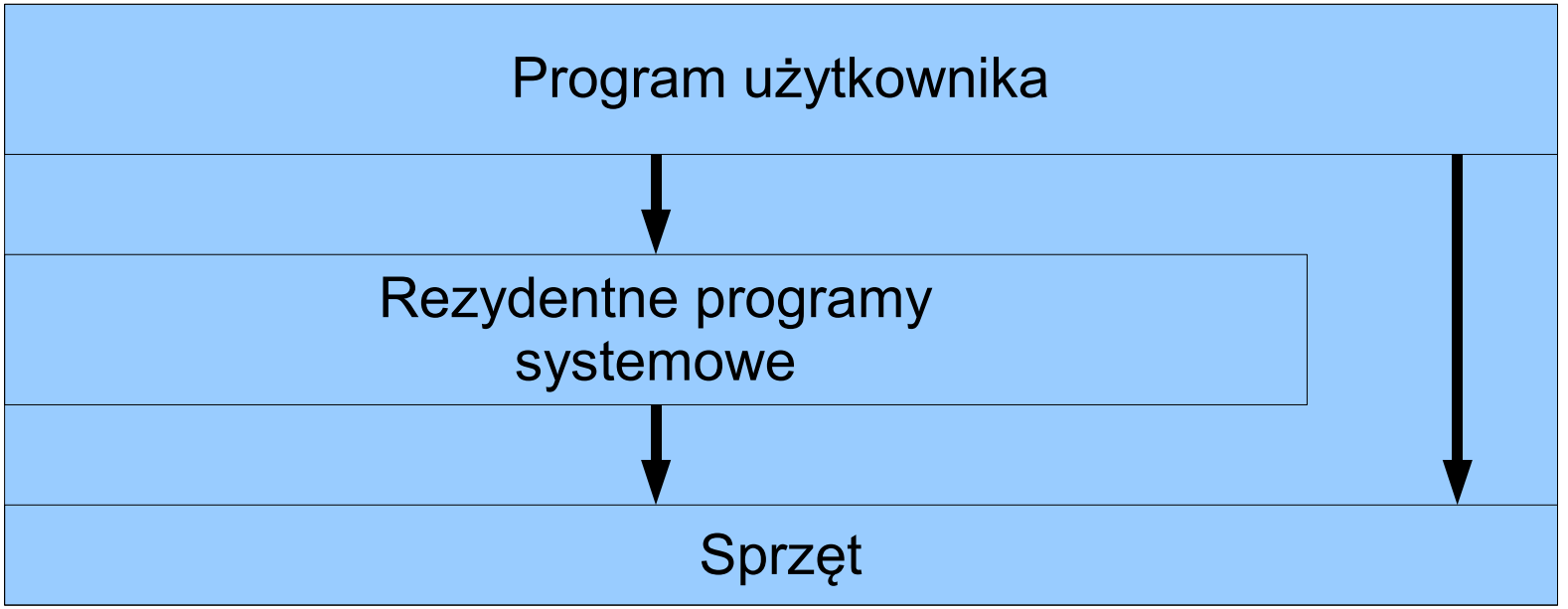


Odmierzanie czasu

- Jedną z najważniejszych i najczęściej spotykanych funkcji systemu czasu rzeczywistego jest generowanie zdarzeń, które mają w określonym czasie uruchomić odpowiednie zadania systemowe.
- Do tego celu wykorzystywane są specjalne obiekty zwane timerami. W celu użycia timera należy go stworzyć, nastawić a następnie uruchomić.
- Podczas tworzenia timera musimy określić rodzaj zdarzeń generowanych przez timer.
- Ustawienie timera polega na przekazaniu mu informacji o planowanym czasie wykonania, sposobie określenia tego czasu i i trybie pracy timera.
- Czas można określić w sposób relatywny (przesunięcie czasowe od chwili bieżącej) bądź absolutny (czas UTC lub lokalny).
- Timery mogą pracować w dwóch trybach: wyzwolenie jednorazowe , wyzwolenie cykliczne.

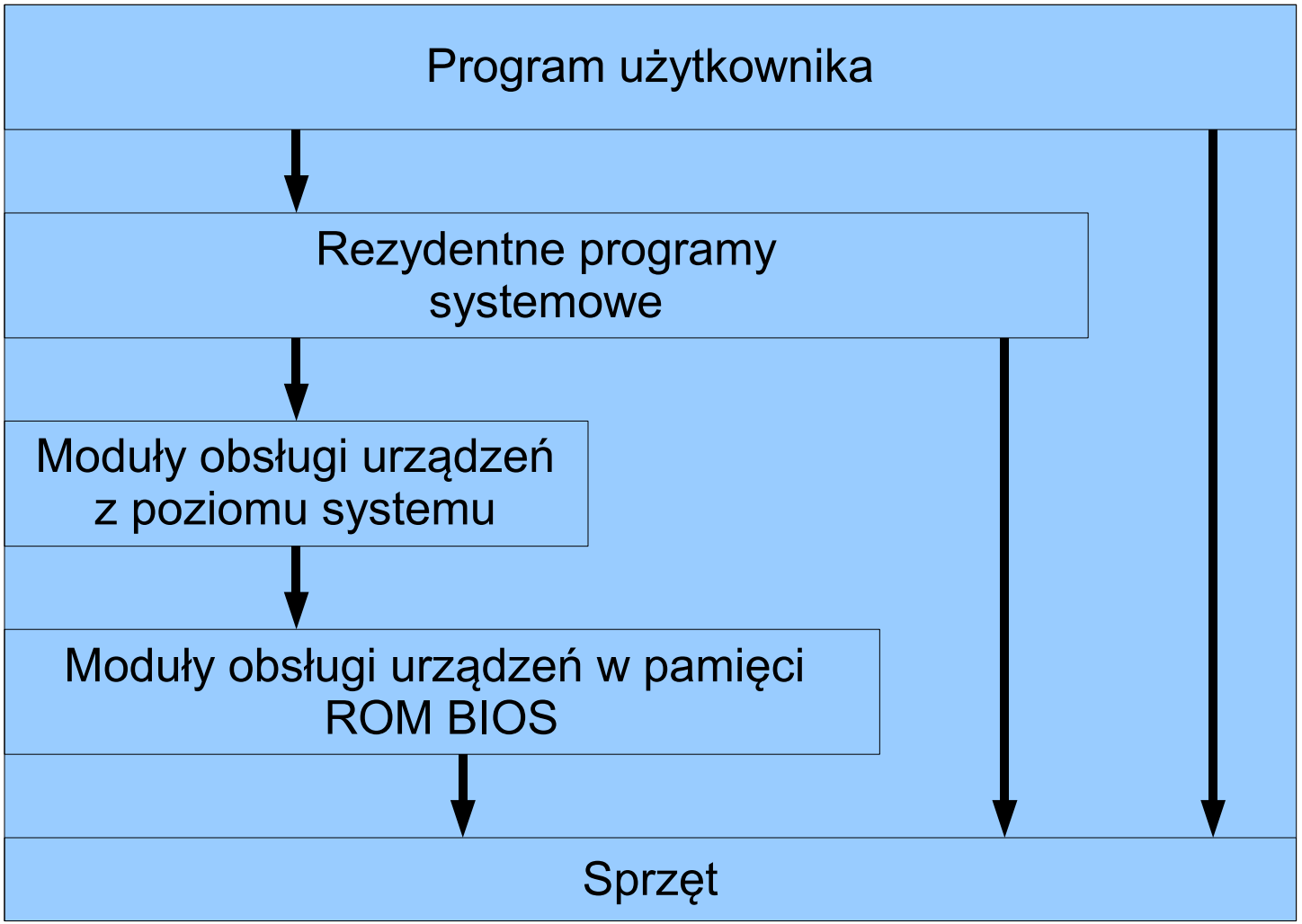
Struktury systemów operacyjnych

- Prosty system operacyjny



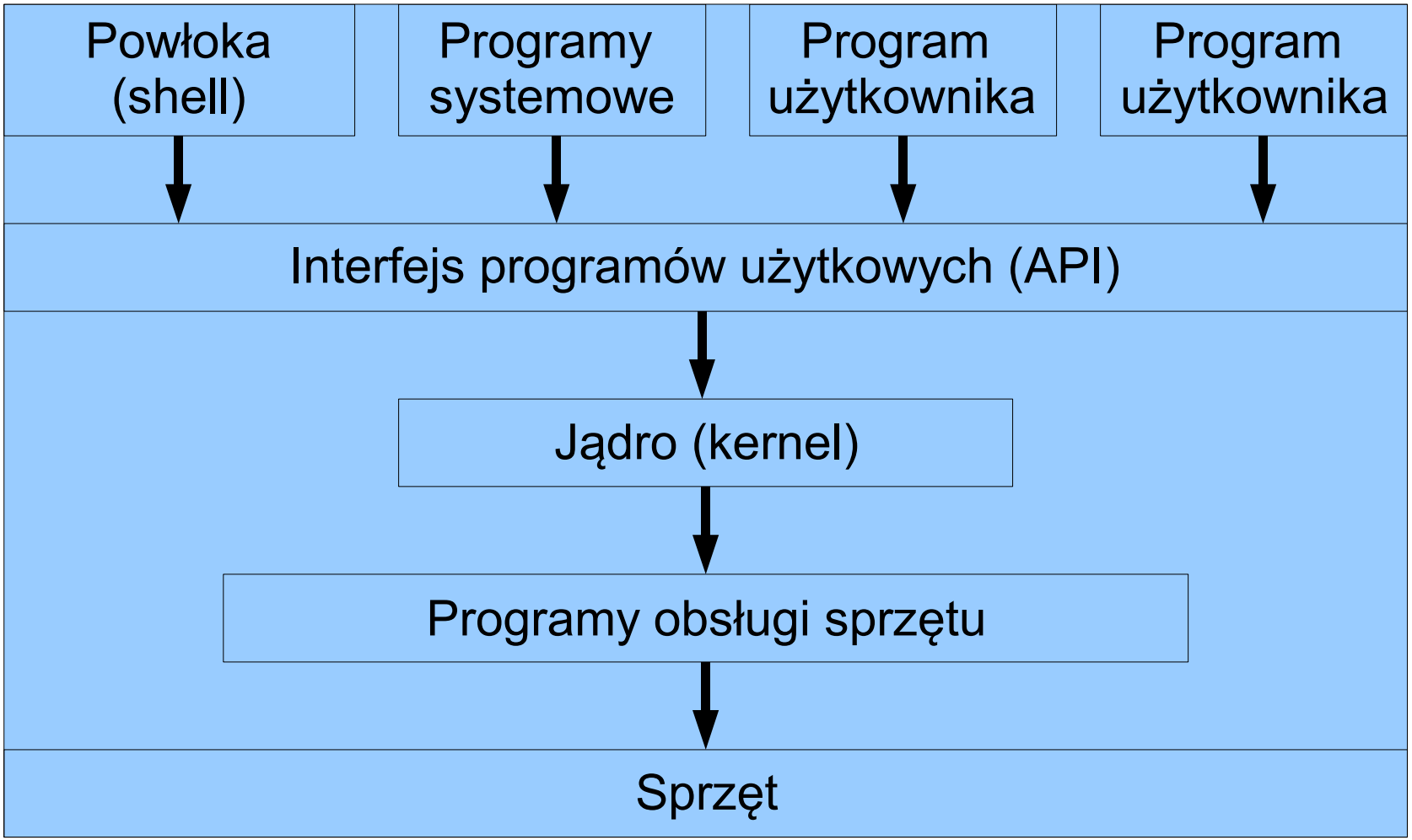
Struktury systemów operacyjnych

- Jednozadaniowy system operacyjny



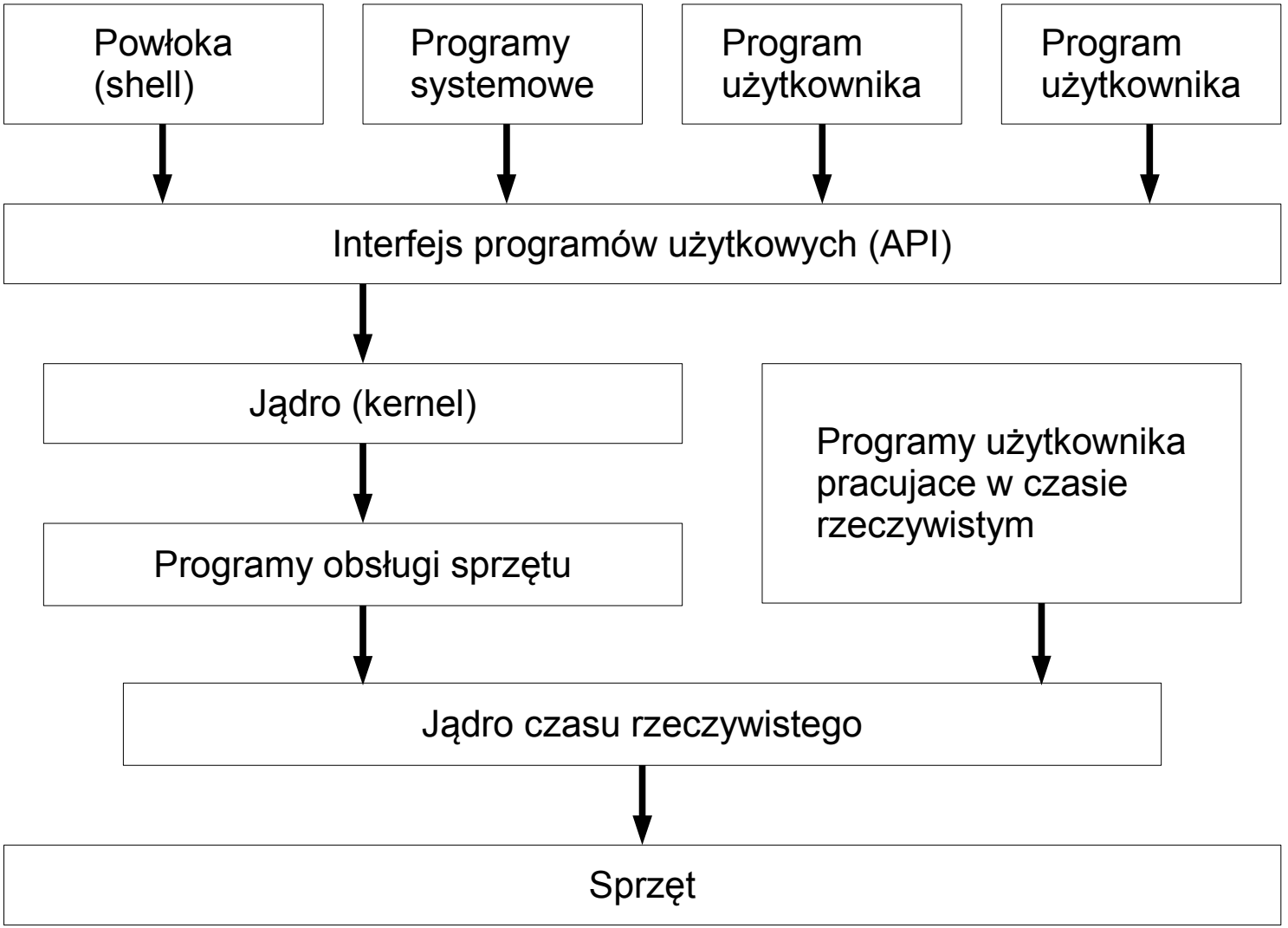
Struktury systemów operacyjnych

- Wielozadaniowy system operacyjny



Struktury systemów operacyjnych

- System operacyjny z rozszerzeniami czasu rzeczywistego



System operacyjny DOS

- Jeden z pierwszych systemów operacyjnych dla platformy PC.
- Jest to prosty system operacyjny umożliwiający w zasadzie obsługę tylko jednego programu użytkownika.
- Zapewnia dostęp do zasobów sprzętowych poprzez odpowiednie sterowniki lub w sposób bezpośredni z wykorzystaniem instrukcji wejścia-wyjścia.
- System ten nie jest dobrą platformą dla nowoczesnych złożonych systemów sterowania, pracujących w czasie rzeczywistym.

System operacyjny DOS

- Jednak prostota obsługi, niska cena i duża popularnośc, przyczyniły się, że jest on jeszcze dość często stosowany w niewielkich systemach, gdzie nie jest wymagana wielozadaniowość.
- W systemie tym istnieje możliwość implementacji procedur pracujących w czasie rzeczywistym poprzez wykorzystanie przerwania zegarowego.
- Wadą systemu jest brak zaimplementowanych procedur obsługi sieci.
- Można to jednak zrealizować poprzez wykorzystanie zewnętrznych bibliotek.

System operacyjny DOS

- Zalety systemu DOS:
  - prostota
  - szybkość implementacji
  - niewielkie rozmiary jądra
  - dostępnośc narzędzi do programowania
  - łatwy dostęp do zasobów sprzętowych
  - duża ilość oprogramowania

System operacyjny DOS

- Wady systemu DOS:
  - brak wielozadaniowości
  - brak wbudowanej obsługi sieci
  - brak wygodnych mechanizmów umożliwiających implementację systemu czasu rzeczywistego
  - brak możliwości zagwarantowania krótkiego czasu reakcji na zdarzenie
  - łatwość naruszenia spójności systemu - brak odporności systemu na błędy

RT Kernel

- Istnieje możliwość obejścia rzęści z tych ograniczeń poprzez zastosowanie nakładki na system operacyjny implementującej część z brakujących mechanizmów, np. RT Kernel.
- Jest to niewielka biblioteka procedur (ok. 16KB kodu i 6kB danych) zapewniająca mechanizmy do efektywnego tworzenia programów pracujących w czasie rzeczywistym.

RT Kernel

- Cechy biblioteki to:
  - liczba zadań limitowana rozmiarem pamięci
  - czas zmiany kontekstu około 2 mikrosekund
  - synchronizacja zadań zegarem czasu rzeczywistego o częstotliwości do 10kHz
  - wywłaszczanie zadań na podstawie ich priorytetów
  - synchronizacja zadań z wykorzystaniem semaforów
  - wymiana informacji między zadaniami z wykorzystaniem skrzynek pocztowych i komunikatów
- Popularność tych dodatkowych bibliotek jest jednak nieduża.

Jądro systemu - rodzaje

- Mikrojądro (ang. microkernel) to rodzaj jądra systemu operacyjnego, które zawiera tylko najbardziej niezbędne elementy, takie jak funkcje zarządzania wątkami, komunikacją międzyprocesową, oraz obsługą przerwań i wyjątków.
- Wszelkie inne zadania, takie jak np. obsługa systemów plików, sieci, sprzętu realizowane są w przestrzeni użytkownika przez osobne serwery.
- Dobrymi przykładami systemów operacyjnych opartych na mikrojądrze są Amoeba, QNX, BeOS , Haiku czy Hurd, mikrojądrami są także (używane w Hurdzie) Mach i L4.
- Firma Microsoft pracuje nad własnym rozwiązaniem tego typu w projekcie Singularity.

Jądro systemu - rodzaje

- Jądro monolityczne - to rodzaj jądra systemu operacyjnego, w którym większa część funkcji jądra zaimplementowana jest w pojedynczym obrazie pamięci, który ładowany jest na stałe do pamięci komputera przez bootloader.
- Zdarza się jednak, że do jądra monolitycznego (zwanego wtedy modularnym) dopisywana jest możliwość ładowania modułów, które jednak nie realizują najbardziej podstawowych funkcji jądra.
- Jądro monolityczne cechuje się wyższą wydajnością niż mikrojądro, zwłaszcza w systemach jednoprocesorowych oraz przy obciążaniu systemu tylko jednym procesem.

Jądro systemu - rodzaje

- Awaria (np. błąd programisty) w dowolnym miejscu jądra monolitycznego może spowodować awarię całego systemu, przez co testowanie tych jąder (szczególnie sterowników urządzeń) jest czasochłonne i skomplikowane, a ewentualne luki mogą często być wykorzystane przez crackerów do łamania zabezpieczeń.
- Tworzenie systemów czasu rzeczywistego przy pomocy jądra monolitycznego, jest zwykle trudniejsze niż w mikrojądrach, z powodu konieczności zagwarantowania ograniczeń czasowych przez wszystkie komponenty (zwykle przy wykorzystaniu wywłaszczania mało ważnych komponentów, oraz unikanie blokad w postaci semaforów).
- Przykładami systemów operacyjnych z jądrem monolitycznym są np. FreeBSD, Linux.

Jądro systemu - rodzaje

- Jądro hybrydowe jest jądrem opartym o zmodyfikowane architektury jądra monolitycznego oraz mikrojądra używanych w systemach operacyjnych.
- W przeciwieństwie do mikrojądra, wszystkie (lub prawie wszystkie) usługi wykonywane są w przestrzeni jądra.
- Podobnie jak w jądrze monolitycznym, nie ma strat w wydajności wywołanych przepływem komunikatów mikrojądra i przełączaniem kontekstu między przestrzenią użytkownika a jądra.
- Jednakże, podobnie jak w jądrach monolitycznych, nie ma korzyści wynikających z umieszczenia usług w przestrzeni użytkownika.
- Z tego typu jądra korzystają m.in. systemy operacyjne z rodziny Windows.

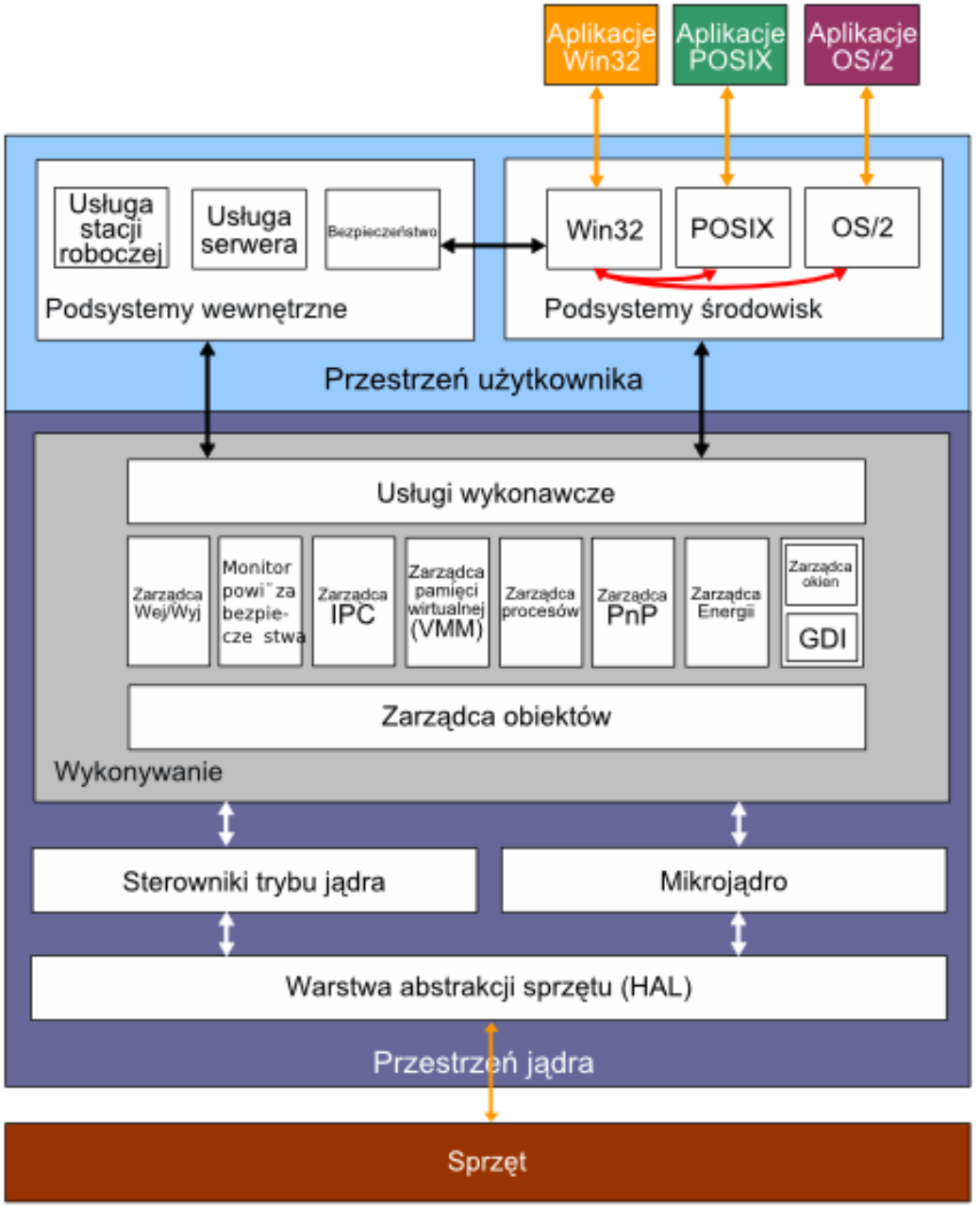
System operacyjny Windows 95, 98, ME

- Systemy rodziny Windows są rozszerzeniem systemu DOS zapewniającym interfejs graficzny oraz możliwość pracy wielozadaniowej.
- Zastosowany mechanizm zawłaszczający powoduje, że system ten nie nadaje się do stosowania w systemach czasu rzeczywistego.
- Związane jest to z tym, że aktualnie wykonywane zadanie może rezerwować zasoby systemu dowolnie długo.
- W efekcie nie jest możliwe zagwarantowanie krótkiego czasu reakcji na zdarzenie czasu rzeczywistego.
- Dodatkowo niepoprawna praca jednej aplikacji może doprowadzić do załamania stabilności systemu.
- Istnieją rozszerzenia systemu umożliwiające pracę w czasie rzeczywistym, jednak nie są one zbyt popularne.

System operacyjny Windows NT, 2000, XP

- Systemy operacyjne z rodziny Windows NT, 2000, XP posiadają nową w pełni 32-bitową architekturę oraz interfejs użytkownika zgodny z systemami Windows 95/98.
- Mechanizmy pracy wielozadaniowej zastały w nich znacznie zmienione, przez co systemy te można wykorzystać do pracy w czasie rzeczywistym.
- System posiada architekturę opartą o jądro hybrydowe.
- Obecnie są to najpopularniejsze systemy stosowane w komputerowych systemach sterowania.

System operacyjny Windows NT, 2000, XP



System operacyjny Windows NT, 2000, XP

- Szeregowanie wątków w Windowsie jest zaimplementowane wewnątrz jądra systemu.
- Nie ma żadnego pojedynczego modułu czy algorytmu odpowiadającego za szeregowanie.
- Aczkolwiek kod rozprzestrzeniony jest w jądrze w zdarzeniach związanych z szeregowaniem.
- Ogólnie wszystkie te metody nazywane są „kernel dispatcher”.

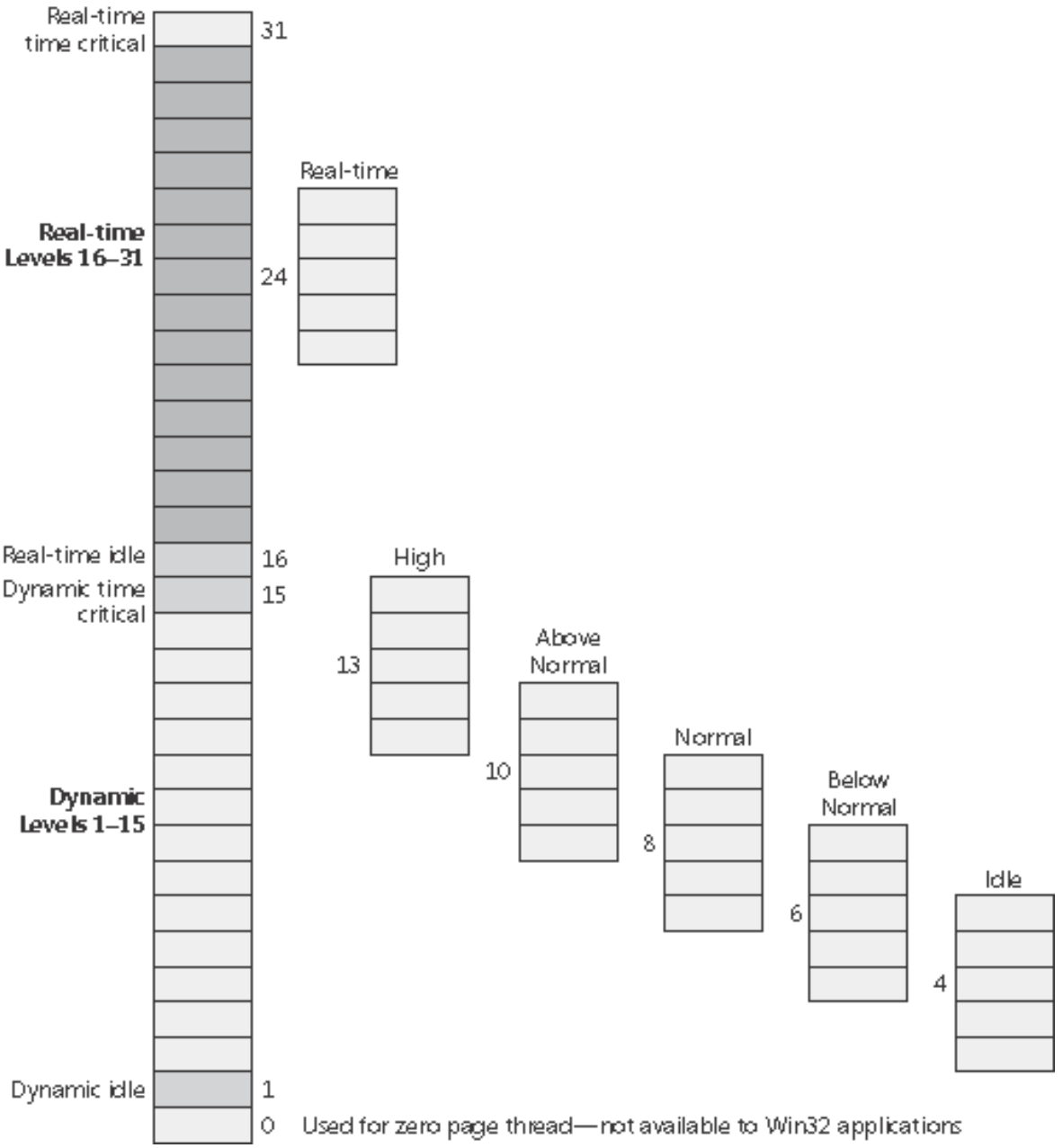
System operacyjny Windows NT, 2000, XP

- Następujące zdarzenia mogą wywołać „dispatching” :
  - Wątek jest gotowy do uruchomienia, np. został utworzony, zakończył status czekania.
  - Wątek oddaje procesor, np. zakończył się, skończył mu się kwant czasu, wszedł w stan czekania.
  - Zmienił się priorytet wątku.
  - Wątek nie może wykonywać się już na danym procesorze, ponieważ zmienił mu się atrybut processor affinity.
- W każdym z tych przypadków, system musi zadecydować jaki wątek powinien być uruchomiony jako następny.
- Gdy już zostanie on wybrany, następuje zmiana kontekstu.

System operacyjny Windows NT, 2000, XP

- Do szeregowania wątków używa się 32 priorytetów, w zakresie od 0 do 31.
- Dzielą się one następująco:
  - Szesnaście poziomów czasu rzeczywistego (16 – 31)
  - Piętnaście dynamicznych poziomów (1 – 15)
  - Jeden poziom systemowy (0)
- Priorytety wątków są tworzone z dwóch różnych perspektyw, Windows API i jądra systemu.
- Najpierw API przyznaje klasę priorytetu procesom, podczas ich tworzenia (Czas rzeczywisty, Wysoki, Powyżej normalnego, Normalny, Poniżej normalnego, Niski),
- Następnie wewnątrz tych klasy istnieją podklasy dla utworzonych przez te procesy wątków (Krytyczny, Wysoki, Powyżej normalnego, Normalny, Poniżej normalnego, Niski).

System operacyjny Windows NT, 2000, XP



System operacyjny Windows NT, 2000, XP

- Proces posiada tylko jedną wartość priorytetu, ale jego wątki już nie.
- Każdy wątek posiada bazowy priorytet (dziedziczony z procesu) i aktualny priorytet.
- To właśnie według aktualnego priorytetu następuje szeregowanie.
- Z różnych powodów Windows może zmienić priorytet wątków w dynamicznym zakresie (1 – 15) przez pewien okres czasu.
- Natomiast nigdy nie zmieni priorytetów wątków w zakresie czasu rzeczywistego.
- Posiadają one zawsze ten sam priorytet bazowy, jak i aktualny.
- Priorytety w dynamicznym zakresie mogą być zmieniane przez użytkownika (o ile posiada prawa do zmiany priorytetów).

System operacyjny Windows NT, 2000, XP

- Stany wątków:
  - Ready(Gotowy) – Wątek jest gotowy do uruchomienia
  - Standby(Oczekiwanie) – Wątek został wybrany do uruchomienia, jeśli zajdą wszystkie potrzebne warunki , nastąpi zmiana dla kontekstu dla tego wątku. Z tego stanu wątek może zostać wywłaszczony.
  - Running(Uruchomiony) – Wątek właśnie jest uruchomiony.
  - Initialized (Tworzenie) – Jest to stan podczas, którego wątek jest tworzony.
  - Terminate(Zakończony) – W tym stanie wątek, znajduje się, gdy zakończy działanie.
  - Waiting (Czekanie) – wątek jest w stanie czekania, np. na jakiś semafor.
  - Transition – wątek jest w tym stanie, gdy jest gotowy do uruchomienia, lecz jądro systemu nie posiada wolnej pamięci. Gdy znajdzie się pamięć, wątek przechodzi do stanu Gotowy.

System operacyjny Windows NT, 2000, XP

- Gdy, wątek jest gotowy do uruchomienia, dostaje procesor na okres kwantu czasu.
- Kwant czasu może być różnej długości, ze względu na kilka czynników:
  - Konfiguracja systemu (długie czy krótkie)
  - Status procesu(Pierwszoplanowy czy w tle)
- Jednakże, proces może nie ukończyć swojego kwantu czasu, gdyż system Windows uwzględnia wywłaszczanie podczas szeregowania wątków.
- Ponadto, wątek może zostać wywłaszczony zanim zacznie wykorzystywać swój kwant czasu.

System operacyjny Windows NT, 2000, XP

- Kwant czasu może być różnej długości.
- W Windowsie XP jest on domyślnie równy dwóm tyknięciom zegara.
- W Windows Server 2003 wynosi on 12 tyknięć zegara.
- Związane to jest z tym aby na systemie serwerowym zminimalizować liczbę zmian kontekstu, dzięki dłuższemu kwantowi czasu, aplikacja serwerowa ma większą szansę na zrealizowanie zapytania klienta.
- Długość tyknięcia zegara zależy od platformy sprzętowej, dla przykładu dla większości jednoprocesorowych maszyn architektury x86 wynosi on około 10 ms, natomiast dla wieloprocesorowych maszyn architektury x86 około 15 ms.

System operacyjny Windows NT, 2000, XP

- Scenariusze szeregowania:
  - Wątek oddaje procesor, np. zawisł na semaforze, operacji wejścia/wyjścia - na jego miejsce wchodzi pierwszy wątek z kolejki o największym priorytecie.
  - Wątek zostaje wywłaszczony – wraca wtedy na początek kolejki o swoim priorytecie.
  - Wątek oddaje procesor, np. skończył mu się kwant czasu, zakończył działanie - wątek wraca na koniec kolejki o swoim priorytecie.

System operacyjny Windows NT, 2000, XP

- Główne cechy to:
  - wywłaszczalność z podziałem czasu
  - zwiększona odporność na błędy sprzętowe i programowe
- Wady systemu:
  - zbyt długi i niestabilny czas obsługi przerwań
  - mała liczba poziomów priorytetów
  - brak odporność na błędy procedur czasu rzeczywistego instalowanych na poziomie jądra systemu
- Powyższe wady spowodowały, że powstały różne rozszerzenia zwiększające wydajność systemu w aplikacjach czasu rzeczywistego (np. RTX API firmy VenturCom) .

Linux

- Historia Linuksa rozpoczęła się w 1991 roku, kiedy to fiński programista, Linus Torvalds poinformował o hobbystycznym tworzeniu przez siebie niedużego, wolnego systemu operacyjnego, przeznaczonego dla procesorów z rodzin i386, oraz i486.
- Linux początkowo działał na platformie i386, lecz później został przeniesiony na wiele innych platform, np. ARM, MIPS, PowerPC, Motorola.
- Jednymi z pierwszych dystrybucji Linuksa były opublikowane w 1993 roku Slackware Linux, czy założony miesiąc później Debian.
- Obecnie najpopularniejsze dystrybucje to Ubuntu, OpenSUSE, Fedora, Debian.

Linux

- Wydania jądra z parzystym drugorzędnym numerem wersji należą do serii stabilnych wydań: 1.0.x, 1.2.x, 2.0.x, 2.2.x, 2.4.x i obecna 2.6.x; wydania z nieparzystym drugorzędnym numerem wersji, np. seria 2.5.x, nazywane są rozwojowymi i nie są zalecane do celów produkcyjnych.
- Linux potrafi uruchamiać programy w formatach a.out oraz ELF.
- Dzięki zaznaczeniu przy kompilacji opcji Kernel support for MISC binaries może również uruchamiać inne programy, np. napisane w Javie poprzez maszynę wirtualną albo przeznaczone dla MS-DOS lub MS Windows poprzez emulatory.

Linux - jądro

- Jądro Linuksa jest w dużym stopniu zgodne ze standardami ANSI i POSIX, obsługuje
  - wielozadaniowość,
  - wielowątkowość, wielobieżność,
  - pamięć wirtualną,
  - biblioteki współdzielone,
  - ładowanie na żądanie,
  - współdzielony kod wykonywalny (ang. copy-on-write),
  - dobre zarządzanie pamięcią
  - i obsługę sieci TCP/IP.

Linux - jądro

- Jest ono jądrem monolitycznym z ładowalnymi modułami.
- Sterowniki urządzeń i rozszerzenia jądra zwykle pracują w trybie ring 0, z pełnym dostępem do sprzętu; nieliczne jednak działają w trybie użytkownika.
- W przeciwieństwie do typowych jąder monolitycznych, sterowniki urządzeń są zwykle kompilowane jako moduły, które można załadować i wyładować na działającym systemie.
- Podobnie, sterowniki mogą być wywłaszczone w określonych warunkach.
- Ta funkcja została dodana w celu poprawnej obsługi przerwań sprzętowych i systemów wieloprocesorowych.

Linux – scheduler w jądrze 2.6

- Posiada on 140 poziomów priorytetów, mniejsza wartość oznacza większy priorytet.
- Priorytety 1-100 odpowiadając procesom czasu rzeczywistego, a pozostałe zwykłym procesom.
- Mamy dwie tablice list priorytetów dla każdego procesora: tablice procesów, którym pozostały jeszcze jakieś kwanty czasu i te, które już swoje kwanty czasu wykorzystały.
- Na listach mamy procesy o tym samym priorytecie, mamy liczniki elementów na danej liście i mapę bitową zajętości. 1 pod i-tym bitem oznacza, że lista o i-tym priorytecie jest nie pusta.
- Tablice te są dostępne poprzez wskaźniki, co czyni operację zmiany epoki niezwykle szybką: po prostu zmieniamy wskaźniki.

Linux – scheduler w jądrze 2.6

- W systemie dostępne są 3 różne strategie szeregowania, jedną dla zwykłych procesów i dwie dla procesów czasu rzeczywistego.
- Każdy proces zwykły ma dwa priorytety – statyczny i dynamiczny.
- Statyczny priorytet zwany “nice” jest z zakresu -20..19, domyślnie jest on ustawiany na 0, ale można go zmienić poprzez wywołanie funkcji systemowej nice().
- Procesy są wykonywane od najwyższych priorytetów, dokładnie jest wykonywany pierwszy z listy procesów o najwyższych priorytecie.
- Gdy wszystkie procesy wykorzystają kwanty czasu rozpoczyna się nowa epoka.

Linux – scheduler w jądrze 2.6

- W zależności od tego czy proces korzysta głównie z procesora, czy też głównie z operacji I/O jest przydzielany mu priorytet dynamiczny.
- Procesy często korzystające z I/O są nieco uprzywilejowane względem procesów korzystających głównie z procesora.
- Ten bonus wyliczany jest na podstawie czasu jaki proces pozostawał w uśpieniu względem maksymalnego czasu oczekiwania.
- Priorytet dynamiczny jest obliczany w momencie budzenia procesu i umieszczania go w Runqueue.

Linux – scheduler w jądrze 2.6

- Linux zapewnia jedynie soft RT.
- Oznacza to iż proces czasu rzeczywistego nie ma gwarancji, że konkretną pracę wykona w konkretnym czasie, czy też że w konkretnym momencie będzie miał on dostęp do procesora.
- Oznacza to jedynie, że w sytuacji gdy jakiś proces czasu rzeczywistego chce pracować, procesora nie dostaną pozostałe procesy.
- Brak możliwości zapewnienia pełnego RT wynika z wielu rzeczy: między innymi z tego iż nie wiemy ile czasu zajmie nam odczytanie strony pamięci (może ona być w cache'u procesora, może być w pamięci, lub trzeba ją będzie ściągnąć z dysku).

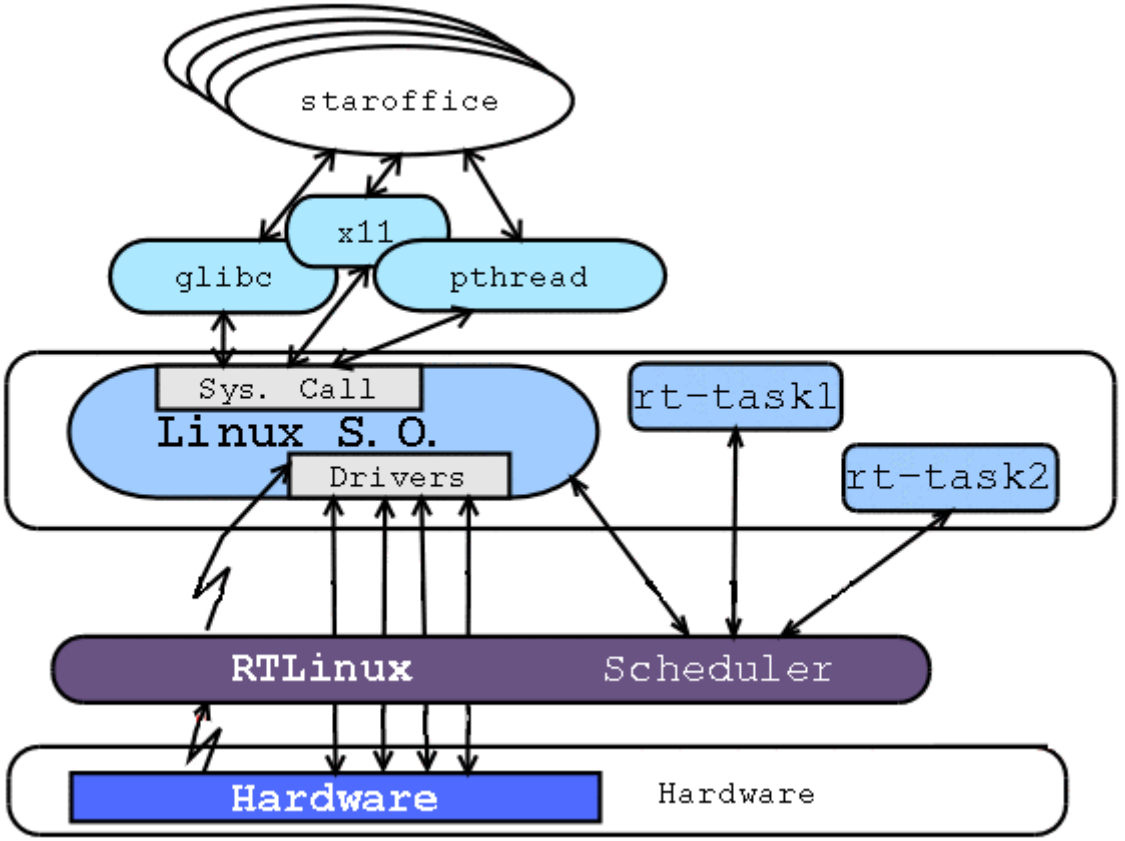
Linux – scheduler w jądrze 2.6

- Linux daje nam 2 strategie szeregowania procesów czasu rzeczywistego.
- Strategia FIFO działa następująco: proces czasu rzeczywistego, który otrzymał procesor, działa dotąd aż zrzeknie się go dobrowolnie, lub pojawi się proces z wyższych priorytetem, który go wydziedziczy. Oczywisty jest przypadek zagłodzenia, gdy dwa procesy czasu rzeczywistego o tym samym priorytecie chcą uzyskać czas procesora, a działa tylko jeden z nich.
- W strategii SCHED_RR procesy mają przydzielane kwanty czasu, działają aż do ich wykorzystania, po czym mają je przyznane ponownie i umieszczane są na końcu kolejki gotowych o danym priorytecie. Oznacza to, że jedynie procesy RT o najwyższym priorytecie dzielą między siebie procesor, podczas gdy pozostałe będą głodzone.

RTLinux

- RTLinux to rygorystyczny (twardy) system operacyjny czasu rzeczywistego, którego cechą charakterystyczną jest to, że współistnieją w nim:
  - jądro czasu rzeczywistego RTCore
  - i jądro Linuksa.
- Pod kontrolą małego jądra czasu rzeczywistego RTCore uruchomione jest jądro Linuksa jako wątek o najniższym priorytecie (idle thread).
- Obecnie istnieją dwie odmiany RTLinuksa: RTLinux/GPL, dostępna na licencji GPL i RTLinuxPro – komercyjna.

RTLinux


staroffice
x11
glibc
pthread
Sys. Call
Linux S. O.
Drivers
rt-task1
rt-task2
RTLinux
Scheduler
Hardware
Hardware

RTLinux - przerwania

- Wirtualny mechanizm przerwań jest kluczowym elementem architektury RTLinuksa.
- Wszystkie przerwania sprzętowe są przechwytywane przez jądro RTCore.
- Jednocześnie emuluje ono kontroler przerwań na potrzeby Linuksa i Linux nadal działa tak, jakby docierały do niego przerwania sprzętowe.
- Każde nadchodzące przerwanie jest przechwytywane przez jądro RTCore.
- Jeśli jest dla niego zainstalowana procedura obsługi czasu rzeczywistego, to jest ona wywoływana.

RTLinux - przerwania

- W przeciwnym razie, o ile nie wykonuje się żadne zadanie krytyczne, przerwanie jest przekazywane do jądra Linuksa. Samodzielne jądro Linuksa czasem blokuje przerwania (np. przy synchronizacji).
- RTCore pamięta ustawiony przez nie stan bitu zezwolenia na przerwania.
- Jeśli jądro Linuksa wyłączy przerwania, to nadchodzące dla niego przerwania są oznaczane jako oczekujące i zostaną emulowane, gdy tylko Linux je odblokuje.

RTLinux – zadania krytyczne

- Zadania krytyczne to programy użytkownika wykonywane w przestrzeni adresowej jądra RTCore, szeregowane przez planistę tego jądra.
- Umieszczenie zadań we wspólnej przestrzeni adresowej bez ochrony pamięci powoduje, że błędy w tych zadaniach mogą doprowadzić do załamania systemu.
- Taka decyzja przynosi jednak następujące korzyści istotne dla zadań czasu rzeczywistego:
  - szybsze przełączanie kontekstu pozbawione narzutu na zmianę rejestru bazowego jednostki zarządzającej pamięcią i czyszczenia rejestrów TLB
  - bezpośrednie dzielenie danych bez skomplikowanej komunikacji międzyprocesowej

RTLinux – zadania krytyczne

- Zadania czasu rzeczywistego umieszczane są w ładowalnych modułach jądra.
- W trakcie inicjacji modułu zadanie są tworzone za pomocą standardowej funkcji pthread_create.
- Zadania mogą być wykonywane okresowo, jak również mogą być zawieszane i budzone z poziomu procedur obsługi przerwań.

RTLinux – szeregowanie

- Program szeregujący jest zaimplementowany jako ładowalny moduł jądra RTCore, więc można napisać własny.
- Rate Monotonic Scheduling - Domyślny planista używa statycznego doboru priorytetów algorytmem RMS, zgodnie z którym im krótszy okres zadania, tym wyższy jego priorytet. Jest to algorytm optymalny w tym sensie, że jeśli zadanie nie jest szeregowalne (nie może być wypełnione w terminie) przez ten algorytm, to nie jest szeregowalne przez żaden algorytm używający statycznych priorytetów.

RTLinux – szeregowanie

- Earliest Deadline First - Obecnie RTLinux jest dystrybuowany razem z dodatkowym modułem implementującym planistę z dynamicznym przydziałem priorytetów. Im bliższy nieprzekraczalny termin wykonania, tym wyższy priorytet. Zaletą tego algorytmu jest 100% ograniczenie szeregowalności, ale wadę stanowi narzut na obliczanie priorytetów.
- Zadania nieokresowe - ani RMS ani EDF nie gwarantują terminowego wypełnienia zadań nieokresowych zwanych sporadycznymi tzn. pojawiających się w dowolnym czasie.
- Algorytmy o nazwach Slot Shifting i Stack Stealing mają na celu lepszą obsługę zadań nieokresowych poprzez wykorzystanie wolnych cykli procesora pomiędzy zadaniami okresowymi.

RTLinux – komunikacja międzyprocesowa

- Tylko zadania, na które nałożono twarde ograniczenia czasowe, działają jako zadania krytyczne.
- Pozostała praca wykonywana jest przez procesy Linuksa m.in. rozruch systemu, większość obsługi urządzeń, sieć, systemy plików itp.
- Jednak jako że wątek jądra Linuksa jest wywłaszczalny, zadania krytyczne nie mogą wywoływać jego funkcji systemowych.
- Niezbędna jest zatem komunikacja.

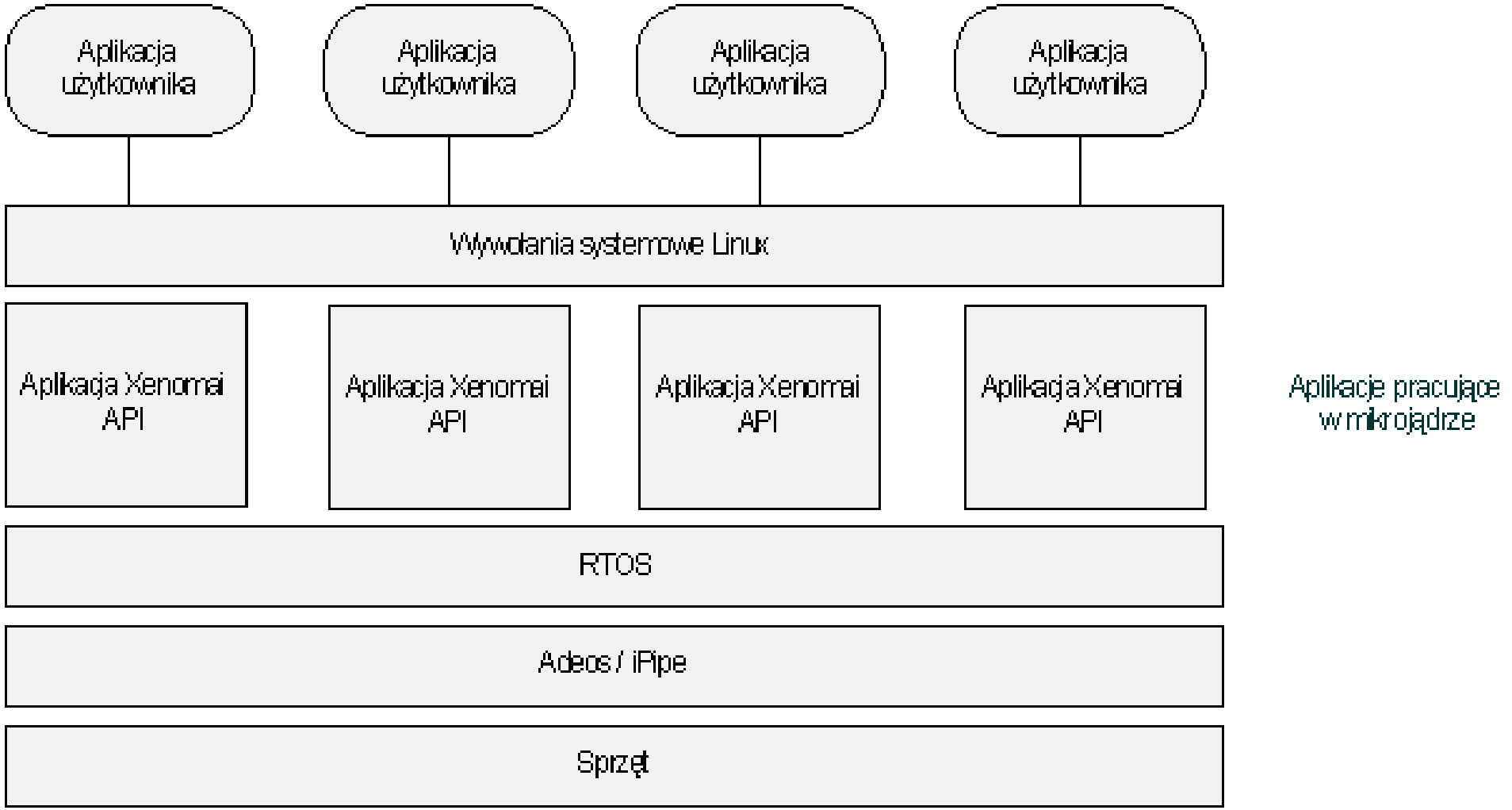
RTLinux – komunikacja międzyprocesowa

- **Kolejki czasu rzeczywistego** RT-FIFO dostępne są w postaci ładowalnego modułu jądra. Są zaalokowane w przestrzeni jądra RTCore. Zadania krytyczne mogą je tworzyć, niszczyć, czytać i pisać. Operacje odczytu i zapisu są po stronie zadań krytycznych niepodzielne i nieblokujące, co rozwiązuje problem odwróconych priorytetów. Procesy Linuksa widzą te kolejki jako zwykłe urządzenia znakowe (/dev/rtf0 itd.), do których mają dostęp poprzez standardowe funkcje POSIXowe.
- **Pamięć dzielona** - niesekwencyjna wymiana danych między procesami Linuksa a zadaniami krytycznymi może przebiegać za pomocą pamięci dzielonej. Różne implementacje RTLinuksa używają różnych funkcji dostępowych do pamięci dzielonej (mmap() w wersji niekomercyjnej i shm_open(), shm_unlink() w wersji komercyjnej), ale zawsze są one zgodne ze standardem POSIX.

Xenomai

- Xenomai jest systemem czasu rzeczywistego współpracującym z jądrem linux'a za pośrednictwem modułu Adeos.
- Dzięki modułowi adeos xenomai przechwytuje zdarzenia występujące w systemie, a następnie decyduje czy maja one zostać wykonane w jądrze linux'a czy też w jądrze pracującym w czasie rzeczywistym Xenomai'a.
- Xenomai ponadto posiada różne API które pozwalają na programowanie aplikacji czasu rzeczywistego.
- W systemie Xenomai zadania czasu rzeczywistego można uruchamiać w przestrzeni użytkownika i w przestrzeni jądra czasu rzeczywistego.

Xenomai


Aplikacja użytkownika
Aplikacja użytkownika
Aplikacja użytkownika
Aplikacja użytkownika
Wywołania systemowe Linux
Aplikacja Xenomai API
Aplikacja Xenomai API
Aplikacja Xenomai API
Aplikacja Xenomai API
Aplikacje pracujące w mikrojądrze
RTOS
Adeos / iPipe
Sprzęt

Xenomai

- Wspierane architektury ppc, blackfin, arm, x86, x86_64, ia64 i ppc64
- wspierane API include POSIX 1003.1b, VxWorks, pSOS+, VRTX and uITRON

QNX neutrino

- Architektura systemu QNX jest modularna.
- Najważniejszym elementem systemu jest mikrojądro o bardzo małym rozmiarze.
- W zależności od potrzeb współpracuje ono z dodatkowymi modułami, takimi jak zarządca procesów, systemu plików, sieci, urządzeń wejścia-wyjścia itp.
- Budowa systemu z gotowych dyskretnych elementów umożliwia wybranie tylko tych modułów, które są w danym przypadku potrzebne.
- Pozwala to na zbudowanie całkowicie w oparciu o QNX sieci przemysłowej składającej się zarówno z dużych komputerów jak i końcowych sterowników wyposażonych jedynie w 256 kB pamięci RAM bez dysku i monitora, sterujących konkretnym urządzeniem.

QNX

- Serce systemu QNX stanowi mikrojadro o bardzo małym rozmiarze (około 10kB).
- Wykonuje ono jedynie 16 funkcji systemowych i zapewnia trzy rodzaje usług:
  - szeregowanie i przełączanie procesów
  - komunikacje miedzy procesami
  - zarządzanie obsługa przerwań

QNX

- Wchodzący w skład jadra egzekutor ma 32 poziomy priorytetów, numerowanych od 0 (najniższy priorytet) do 31.
- Z tego dla programów uzytkownika dostepne sa wartosci 0-19.
- Administrator systemu (użytkownik root) może uruchamiać procesy nadając im również wyższe priorytety: 20-29.
- Priorytet nadany procesowi podczas tworzenia go można później zmienić za pomocą odpowiednich funkcji systemowych.
- Istnieje również możliwość pracy procesu z tzw. płynnym priorytetem, co wykorzystuje miedzy innymi zarzadca systemu plików.

QNX szeregowanie

- Procesy znajdujące sie na tym samym poziomie priorytetu mogą być szeregowane zgodnie z jednym z trzech różnych algorytmów:
  - FIFO - do wykonania wybiera się najdłużej oczekujący gotowy proces. Wybrany proces wykonuje się aż do zakończenia lub zawieszenia.
  - Karuzelowy - procesy wybiera się do wykonania kolejno, przy czym każdy z nich otrzymuje ograniczony kwant czasu (slice). Jeżeli wykonywany proces nie zakończy się lub nie zawiesi wcześniej, to w chwili wyczerpania kwantu system operacyjny wywłaszcza go i podejmuje wykonanie następnego procesu.

QNX szeregowanie

- Procesy znajdujące się na tym samym poziomie priorytetu mogą być szeregowane zgodnie z jednym z trzech różnych algorytmów:
  - Adaptacyjny (dostosowujacy sie) - algorytm znany z UNIX'a. Procesy wybiera się kolejno, podobnie jak w algorytmie karuzelowym. Po wyczerpaniu kwantu czasu system operacyjny wywłaszcza wykonywany proces i obniża jego priorytet o 1. Jeżeli wykonanie procesu o obniżonym priorytecie nie zostanie podjęte w ciągu 2 sekund, jego priorytet podnosi się o 1 (ale nigdy powyżej oryginalnego priorytetu). Proces zawieszony odzyskuje natychmiast swój oryginalny priorytet..

QNX komunikacja międzyzadaniowa

- Na poziomie jadra zaimplementowane zostały następujące mechanizmy komunikacji miedzyzadaniowej:
  - przekazywanie komunikatów,
  - sygnały
  - pełnomocnicy (proxy).
- Mechanizm przesyłania komunikatów został zaprojektowany w ten sposób, aby nie zmuszać jadra do przechowywania dużej liczby wysłanych ale nie odebranych jeszcze komunikatów, oraz aby nie wymagał wielokrotnego kopiowania komunikatów przed dostarczeniem ich do adresatów.
- Jadro zajmuje się również wywoływaniem procedur obsługi przerwań dla procesów, które tego zażądały.

QNX komunikacja międzyzadaniowa

- Wszelkie pozostałe usługi systemowe realizowane są przez współpracujących z jadrem odpowiednich zarządców.
- Idea tego podejścia polega na udostępnieniu przez jadro jedynie zestawu podstawowych funkcji, w oparciu o które budowane są następnie funkcje usługowe wyższego poziomu.
- Procesy zarządców zasobów mogą być dynamicznie uruchamiane oraz usuwane w trakcie pracy systemu. W danej chwili mogą pracować tylko te, które są potrzebne.
- Jeżeli np. dany węzeł korzysta z lokalnego systemu plików (dyskietki) tylko sporadycznie, zarządca systemu plików możenie być stale obecny w pamięci, a jedynie uruchamiany w razie koniecznosci i niszczony po wykonaniu operacji wejscia-wyjscia w celu zwolnienia zajmowanej przez niego pamieci.

Komunikacja procesów

- Jadro systemu QNX realizuje cztery mechanizmy komunikacji procesów:
  - synchroniczny przekaz wiadomosci (message) podczas spotkania, synchronizacje
  - za pomoca semaforów (semaphore), asynchroniczny przekaz depozytów
  - (stałych wiadomosci) poprzez procesy depozytowe (proxy)
  - sygnalizacje zdarzen (signal).
- Wszystkie mechanizmy - oprócz semaforów - mogą być wykorzystane do komunikacji procesów wykonywanych w różnych węzłach sieci.
- Dodatkowo, odpowiednie procesy systemowe umożliwiają buforowane przekazywanie danych miedzy procesami poprzez potoki (pipe) i kolejki (FIFO) oraz kolejki wiadomosci (message queue).

QNX - zastosowania

- Przykładowe zastosowania systemu QNX to:
  - sterowanie elektronika atomowa,
  - kierowanie systemem bezpieczeństwa nowojorskiej giełdy,
  - monitorowanie ruchu w tunelu pod kanałem La Manche,
  - testowanie silników odrzutowych.