Real-Time Linux

Wprowadzenie

Co to jest Real-Time ?

Zdolność systemu do udzielenia odpowiedzi (prawidłowej) w z góry określonym czasie.

A więc od systemu czasu rzeczywistego będziemy wymagali, aby reagował na czynniki zewnętrzne w czasie który jesteśmy w stanie przewidzieć.

Czy zawsze ?

Rzeczywiście, wymaganie „punktualności’’ jest podstawowym obok poprawności wykonywanych działań (obliczeń) wymaganiem stawianym systemom real-time’owym.

W pewnych sytuacjach są dopuszczalne jednak niewielkie!!! odstępstwa.

Soft Real-Time

W niektórych zastosowaniach nie jest konieczne, aby system odpowiadał zawsze w ściśle określonym czasie. Może się zdarzyć, jednak nie zbyt często, że system nie odpowie na czas. W tym przypadku nie jest to uznawane za błąd, pod warunkiem, że opóźnienie nie będzie zbyt długie.

Systemy znajdujące zastosowanie w takich sytuacjach określamy mianem Soft Real-Time.

A co z Hard?

Systemy z grupy Hard Real-Time z kolei muszą ZAPEWNIĆ spełnienie wymogów czasowych i to za każdym razem. Przekroczenie dopuszczalnego czasu reakcji na zdarzenie („Deadline'') jest w tych systemach uważane za poważny błąd.

Kto tego potrzebuje ?

Każdy. Przykłady zastosowań systemów czasu rzeczywistego znajdziemy praktycznie w każdej dziedzinie naszego życia.

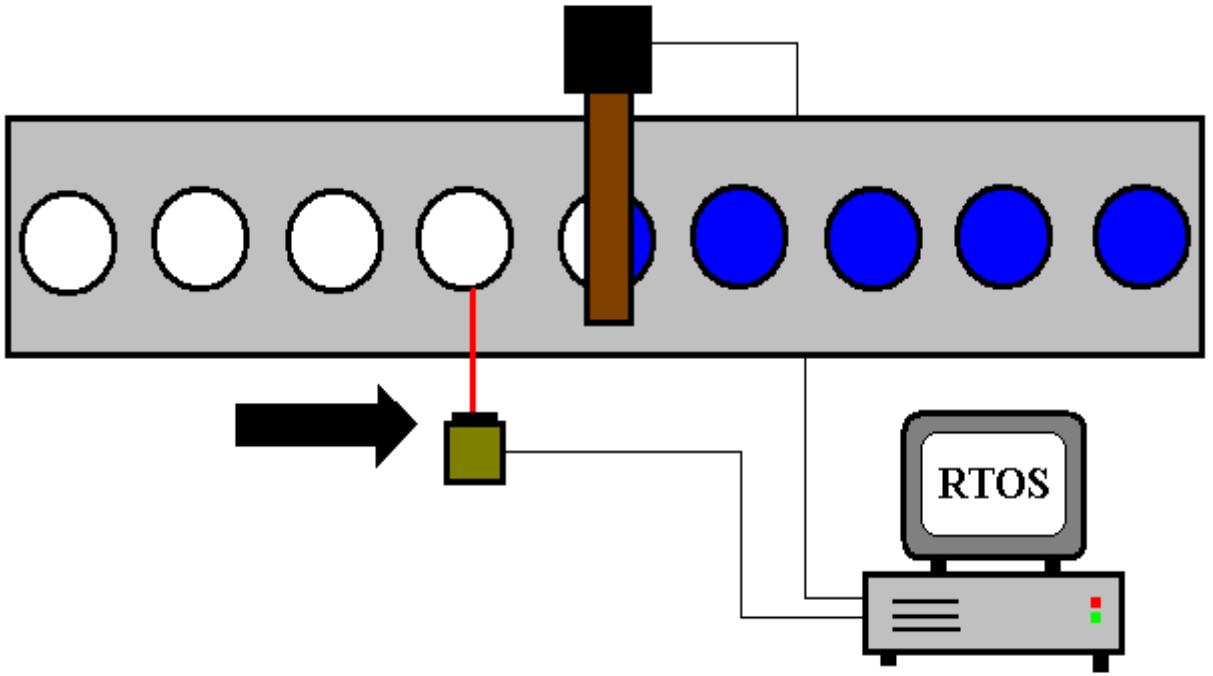
- systemy przetwarzania audio-video
- telekomunikacja
- sterowanie sprzętem w fabrykach
- loty kosmiczne
- i wiele, wiele innych

Soft vs Hard

W większości przypadków zastosowanie systemu typu Soft RT jest wystarczające.

Np. systemy obsługujące bankomaty. Opóźnienia rzędu kilku sekund nie mają większego znaczenia. W przypadku jednak gdy przekroczenie czasu reakcji może być okupione dużymi stratami (np.obsługa maszyn w fabrykach czy sterowanie aparaturą monitorującą życie pacjentów w szpitalu) konieczne staje się zastosowanie systemów Hard RT.

Przykład dydaktyczny



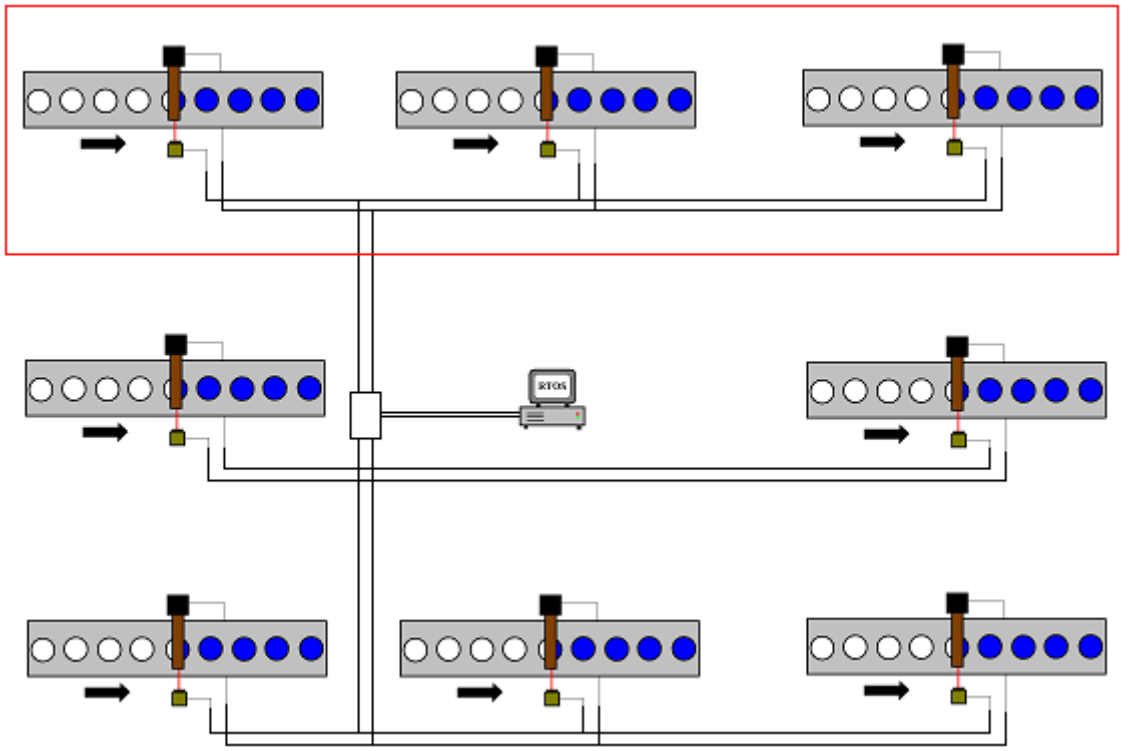
Sterowanie urządzeniem malującym

Problemy z malowaniem

- nie można zacząć zbyt wcześnie
- nie można zacząć zbyt późno
- nie można zakończyć zbyt wcześnie
- nie można zakończyć zbyt późno

Czyli w RTOS nie chodzi tylko o wykonanie zadania przed „Deadline’’, ale przede wszystkim o wykonanie zadania w dokładnym przedziale czasowym.

Wiele taśm



Niezależnie od liczby procesów obsługujących taśmy system zawsze musi spełniać wymagania czasowe. Okresowe wyłączenie niektórych taśm nie może wpływać na działanie pozostałych.

Pozostałe wymagania

- deterministyczny czas wykonania (już było)
- wieloprogramowość i zarządzanie wykonaniem procesów
- komunikacja międzyprocesowa
- dynamiczna alokacja pamięci
- zarządzanie urządzeń I/O
- zegary

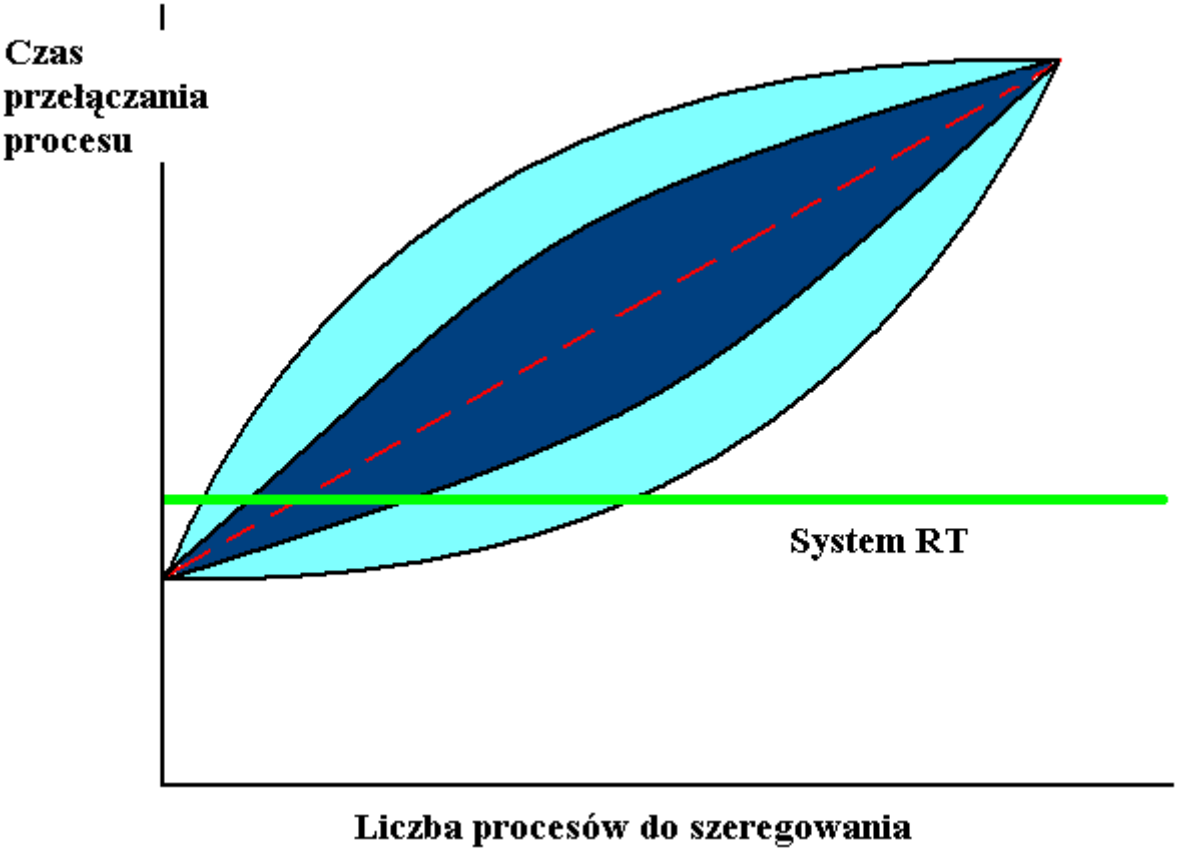
Czy Linux jest RT ?

- ✓ jest systemem wieloprogramowym
- ✓ posiada mechanizmy komunikacji międzyprocesowej (IPC)
- ✓ umożliwia dynamiczną alokację pamięci
- ✓ umożliwia zarządzanie sprzętem I/O
- ✓ dostarcza funkcje wspierające odmierzanie czasu
- ✗ nie umożliwia dokładnej kontroli czasu wykonania, niezależności od liczby procesów

Wiele taśm, a `schedule()`

Aby niezależność od liczby procesów mogła być osiągnięta funkcja `schedule()` w jądrze linux'a musiałaby działać w czasie stałym. W rzeczywistości jest to *`O(n)`*. Nawet dla procesów czasu rzeczywistego funkcja `schedule()` przegląda całą kolejkę `runqueue` i w ten sposób wybiera proces do wykonania.

Przełączanie procesów



Wywłaszczanie w trybie jądra

Rozważmy scenariusz (taśma c.d.):

1. proces o niskim priorytecie wykonuje funkcję systemową
2. w kolejce procesów gotowych pojawia się proces o wysokim priorytecie który jest sterowany czujnikiem na taśmie w fabryce
3. proces o niskim priorytecie wykonuje funkcję trybu jądra i wcale mu się nie spieszy
4. element nie zostaje pomalowany ponieważ proces obsługujący maszynę nadal czeka w kolejce
5. nikt nigdy więcej nie użyje Linux'a jako systemu RT ☺

Przerwanie zegarowe

W standardowym jądrze Linux'a przerwanie zegarowe jest wywoływane co ok. 10 ms (100 Hz).

Oznacza to, że jeżeli nasza aplikacja, której nakażemy zasnąć na 15 ms zostanie tak naprawdę obudzona dopiero po czasie ok. 20 ms. Dla niektórych zastosowań taka dokładność jest niezadowalająca. W takiej sytuacji trzeba szukać innych rozwiązań.

Jak można to rozwiązać ?

- uniezależnić scheduler od liczby procesów (scheduler przynajmniej dla procesów czasu rzeczywistego `SCHED_FIFO` i `SCHED_RR` powinien działać w czasie *`O(1)`*
- wprowadzić wywłaszczanie procesów w trybie jądra
- wprowadzenie dodatkowych (dokładniejszych) mechanizmów odliczania czasu
- przebudowanie całego jądra systemu (sub-kernel)

Wprowadzenie schedulera *O(1)*

Można poprawić scheduler dla procesów czasu rzeczywistego tak, aby wybór procesu odbywał się w czasie stałym (patche na standardowe jądro Linux'a są dostępne w sieci).

Okazuje się jednak, że opóźnienia związane z działaniem `schedule()` nie są tak duże (rzędu mikro sekund) zwłaszcza gdy nasz system będzie pracował jako system Soft RT.

Znacznie większe opóźnienia są spowodowane przez odkładanie wywołania `schedule()`bo np. jakiś proces jest teraz w trybie jądra.

Jądro wyłaszczalne

Poprzez szereg badań i obserwacji zauważono, że można znacznie zwiększyć interakcyjność systemu Linux gdy istnieje możliwość wywłaszczania procesów pracujących w trybie jądra (przykład).

Nie zawsze jednak możemy dopuścić do tego, aby kod jądra był wykonywany przez procesy w dowolnym przeplocie (dostęp do niektórych krytycznych danych). Nie wszystkie funkcje systemowe są wielowejściowe.

Dwa podejścia do sprawy

Na przestrzeni kilku ostatnich lat wypracowano dwa rozwiązania problemu wywłaszczania procesu w trybie jądra:

- jądro w pełni wywłaszczalne (preemptible)
- wywłaszczanie tylko w określonych miejscach (preemption points – low latency)

Obydwa podejścia mają swoich zwolenników i przeciwników. Praktyka natomiast pokazuje, że najlepsze wyniki uzyskuje się łącząc obydwa rozwiązania w jedno.

Preemptible kernel

Pierwsze podejście zakłada, że dopuszczamy wywłaszczanie procesu działającego w trybie jądra poza pewnymi miejscami kodu (sekcje krytyczne) gdzie byłoby to niebezpieczne.

Istnieje grupa łatek na standardowe jądro (preemption patches) które umożliwiają korzystanie z wywłaszczania.

Aby unikać wywłaszczania w sekcjach krytycznych przerabia się funkcje działające na `spin lock'`ach tak, aby blokowały wywłaszczanie na czas gdy proces posiada blokadę. W momencie kiedy blokada jest zwalniana sprawdzane jest czy nie zdarzyło się coś co może wymagać uruchomienia `schedule()` i jeżeli tak było (ustawiona flaga `need_resched` w `task_struct` jest wywoływane `schedule()`).

Low Latency

Termin ten określa drugie podejście do sprawy wywłaszczania procesów pracujących w trybie jądra. Zastosowano tu sprytną sztuczkę. W kodzie jądra są wstawiane fragmenty kodu (preemption points) mające za zadanie dać szanse funkcji `schedule()` na zamianę wykonywanego procesu na inny.

Preemption point:

```
if (current->need_resched) schedule();
```

Punkty te oczywiście nie są losowe. Miejsca takie dobiera się poprzez długie, doświadczalne analizy działania procesów. Poza tym patch'e low latency wprowadzają jeszcze jedno ciekawe rozwiązanie.

Lock Breaking

Załóżmy, sytuację w której przeszukujemy listę pewnych elementów. Nasz proces posiada blokadę związaną z tą listą tak, aby nikt nam się nie wtrącał. Jeżeli lista jest długa to przeglądanie będzie trwało długo.

Jeżeli w trybie jądra proces przeszukuje jakąś listę (zwykle zabezpieczoną `spin lock`'iem) to nie moglibyśmy go wywłaszczyć. Dzięki zastosowaniu *Lock Breaking* możemy przerwać przeszukiwanie listy jeżeli uznamy, że trwa to zbyt długo. Wówczas musimy oddać `spin lock` i sprawdzić czy nikt nie próbuje wywołać `schedule()`. Jeżeli możemy kontynuować działanie to pobieramy blokadę i szukamy dalej. Jednak jeżeli `need_resched` = 1 to musimy dać szanse innym.

W tym przypadku musimy uważać, aby następnym razem zacząć przeszukiwać listę od początku !!!

Dokładne odmierzanie czasu

Dokładne mierzenie czasu jest bardzo ważne dla systemu czasu rzeczywistego. Dla niektórych zastosowań standardowa dokładność 10 ms oferowana przez funkcje jądra jest wystarczająca, w innych jest to zdecydowanie za mało.

Linux korzysta z zegara programowalnego do liczenia czasu. Aby zwiększyć dokładność można go przeprogramować, ale to spowoduje zwiększenie narzutu na obsługę przerwania zegarowego. Trzeba poszukać lepszego rozwiązania.

TSC (*Time Stamp Counter*)

Procesory x86 od *Pentium* począwszy, są wyposażone w 64-bitowy rejestr znacznika czasu TSC zwiększany przy każdym cyklu zegara. Rejestr ten można odczytać za pomocą instrukcji `rdtsc`.

Oznacza to, że na procesorze o częstotliwości zegara 1GHz można uzyskać dokładność pomiaru rzędu 1 ns!!! (64-bity starczy na ok. 600 lat).

Wiele implementacji Real-Time Linux'a wykorzystuje **TSC** do odmierzania czasu,

uzyskując w ten sposób rewelacyjną dokładność.

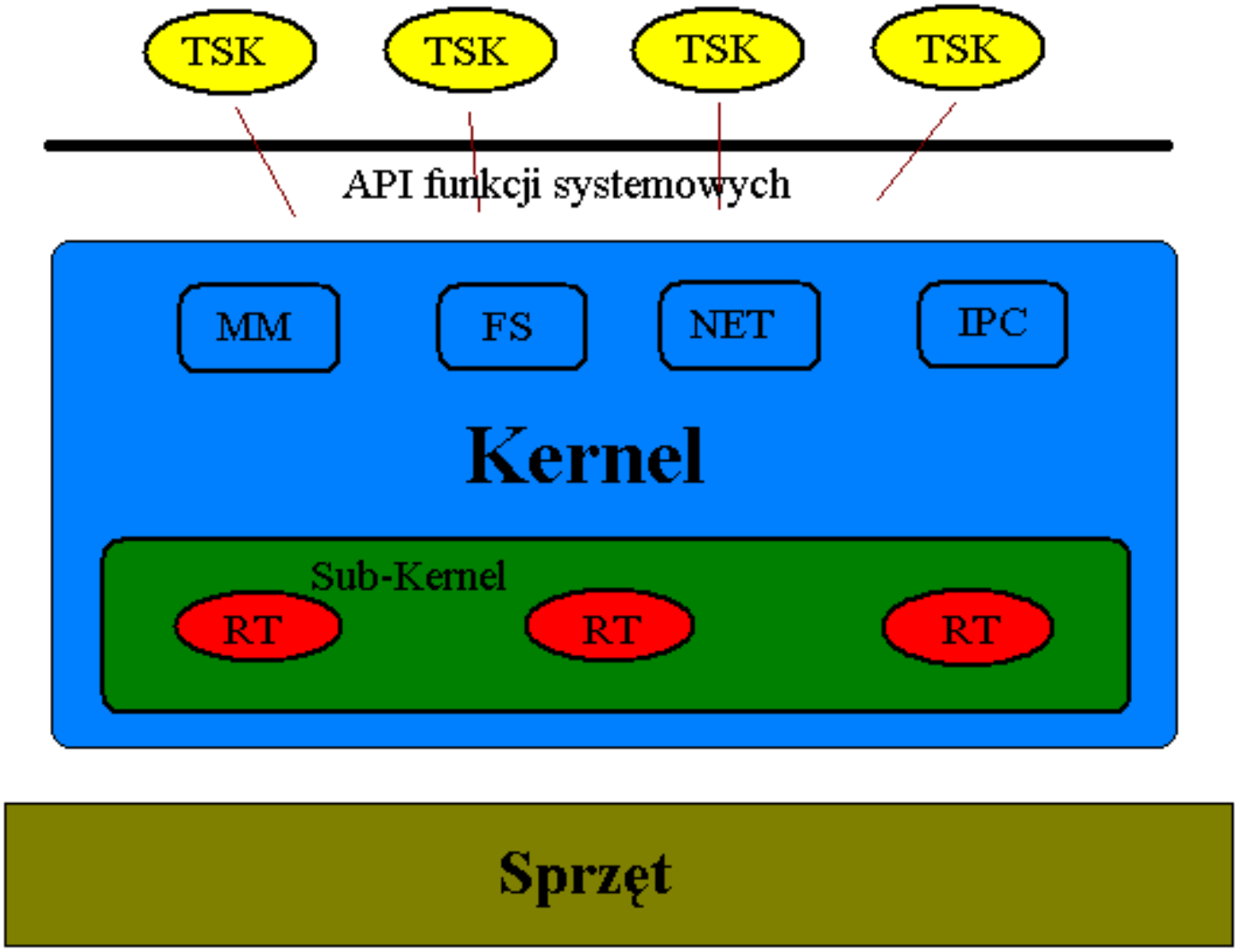
Coś nowego – **Sub-Kernels**

W systemach czasu rzeczywistego, zwłaszcza tych typu Hard, konieczne jest odpowiadanie na przerwania sprzętowe w deterministycznym czasie.

Sprytnym wydaje się więc stworzenie mini-systemu operacyjnego, zajmującego się obsługą przerwań. Warto by było jednak zachować pełną funkcjonalność Linux'a.

Rozwiązaniem jest system w którym jądro Linux'a traktowane jest jako proces RT o najniższym priorytecie.

Sub-Kernel i Kernel



Rola i zadania Sub-Kernel

Sub-Kernel jest małym systemem operacyjnym. Musi więc udostępniać procesom pewne usługi.

- zarządzanie procesami czasu rzeczywistego oparte na priorytetach
- funkcje obsługi przerwań
- funkcje komunikacji z procesami Linux'a

Czyli system ten przejmuje na siebie wszystkie zadania krytyczne dla systemu RT. Aby nie tracić jednak wszystkich zalet jądra Linux'a jest ono utrzymywane, ale jego priorytet nie może być wyższy niż procesy Real-Time.

W ten sposób są konstruowane systemy RT o największych wymaganiach (**`RTLinux,RTAI`**).

Inne pomysły

- różne wersje schedulera ładowane jako moduły jądra
- funkcje obsługi przerwań ustalane z poziomu użytkownika
- API do rezerwowania sprzętu przez procesy
- proces o wyższym priorytecie może odebrać siłą zasób procesowi o niższym priorytecie

  (ma to zapobiegać tzw. **priority inversion**)

Gwarancja spełnienia wymagań

Aby ktoś mógł zagwarantować, że system spełni wymagania konieczne jest przebadanie wszystkich ścieżek wykonania procesów. Samo testowanie nie wystarczy ponieważ nie jesteśmy w stanie przewidzieć wszystkich warunków pracy systemu. Aby system **gwarantował** konieczne jest skonstruowanie matematycznej formuły i jej dowód.

Ze względu na liczbę możliwych wykonań i trudność analizy nikt tego jednak nie robi.

Z tego też względu sprzedawcy systemów RT nie dają gwarancji na spełnienie wymagań, a jedynie stwierdzają, że przekroczenie jakiegoś czasu jest uważane w systemie za błąd.

Informacje o RT w sieci

- http://linuxdevices.com - ogromny zbiór artykułów, opinii, komentarzy
- strony producentów systemów Real-Time