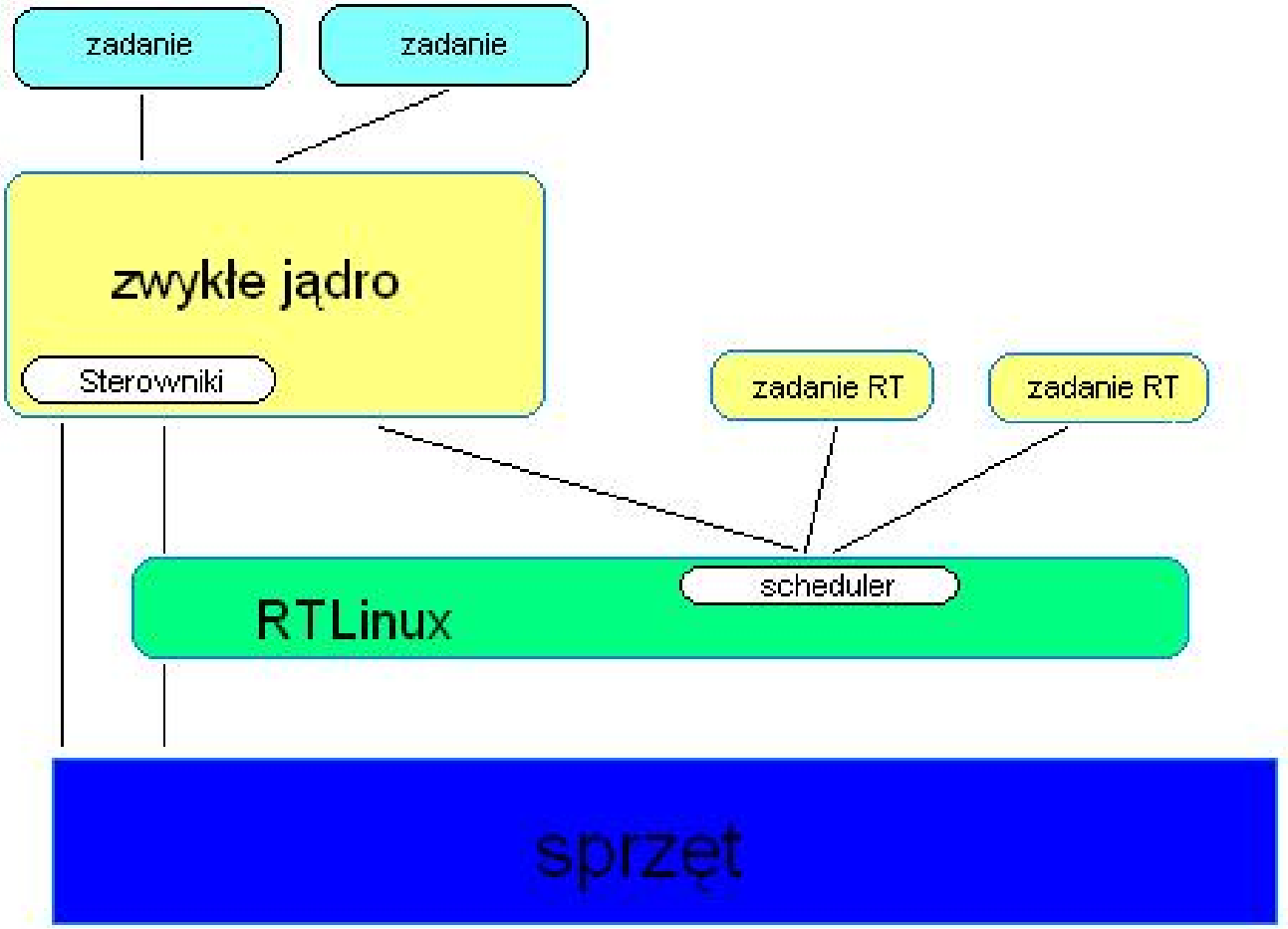
Prezentacja systemu RTLinux

Podstawowe założenia

RTLinux jest system o twardych ograniczeniach czasowych (*hard real-time).* Inspiracją dla twórców RTLinux'a była architektura systemu MERT. W zamierzeniach RTLinux ma mieć możliwość uruchamiania procesów czasu rzeczywistego oraz zwykłych aplikacji. Intencją było stworzenie systemu, w którym system czasu rzeczywistego i system ogólnego przeznaczenia współistnieją razem.

Architektura systemu

- RTLinux oddziela mechanizmy systemu operacyjnego czasu rzeczywistego od systemu operacyjnego ogólnego zastosowania.
- Zwykłe jądro Linux'a działa jako zadanie pod kontrolą systemu operacyjnego czasu rzeczywistego.
- RTLinux emuluje mechanizm kontroli przerwań, uniemożliwiając ich blokowanie przez zwykłe jądro linux'a.
- Zwykły Linux zajmuje się: startem systemu, inicjalizacją urządzeń, ładowaniem modułów, systemem plików i dynamicznym przydzielaniem zasobów.
- RTLinux dostarcza bezpośredniego dostępu do sprzętu dla zadań RT, szeregowania zadań RT, mechanizmów odmierzania czasu i technik komunikacji międzyprocesowej.



Architektura systemu RTLinux

Wirtualny system przerwań

W RTLinux'ie istnieje programowa warstwa emulacji pomiędzy jądrem Linux'a a układem kontroli przerwań. Wszystkie wystąpienia instrukcji *cli, sti, iret* w kodzie jądra są zamienione odpowiednio na makra: *S_CLI, S_STI, S_IRET*. Dzięki temu RTLinux jest w stanie wykryć próby włączania i wyłączania przerwań w zwykłym jądrze. Mechanizm ten uniemożliwia zablokowanie przerwań sprzętowych przez zwykłe jądro. Po próbie wyłączenia przerwań (wykonaniu makra *S_CLI)* przerwania obsługiwane przez Linux'a są zaznaczane jako oczekujące. Po wykonaniu makra S_STI (włączeniu przerwań) procedury obsługi przerwań oczekujących są wykonywane.

Makro *S_CLI*

Jego wywołanie, zamiast rzeczywistego wyłączenia przerwań, zeruje odpowiednią zmienną w emulatorze przerwań. Fakt wystąpienia przerwania w przypadku gdy zmienna ta jest wyzerowana jest zapamiętywany, natomiast procedura obsługi tego przerwania ustalona przez jądro Linux'a nie jest wykonywana.

Makro *S_STI*

Jego wywołanie powoduje włączenie przerwań. Przygotowuje ono stos procesora tak jak w przypadku wywołania przerwania, tzn. odkłada na niego flagi procesora, rejestr segmentowy jądra i adres powrotu po czym wywołuje makro *S_IRET*

Makro *S_IRET*

Makro wykonuje całą pracę emulatora przerwań. Przeszukuje zmienną, na której bitowo zapamiętane są oczekujące przerwania. Ustawiony bit powoduje wywołanie odpowiedniej procedury obsługi przerwania w jądrze Linux'a. Wykonanie makra *S_IRET* w tej procedurze powoduje odnalezienie następnego oczekującego przerwania. Po wyczerpaniu listy przerwań oczekujących wykonywana jest instrukcja *iret.*

Zadania czasu rzeczywistego

Zadania czasu rzeczywistego RTLinux'a wykorzystują mechanizm modułów ładowalnych. Wykonywane są we wspólnej przestrzeni adresowej (przestrzeni jądra), z maksymalnym poziomem uprzywilejowania i bezpośrednim dostępem do urządzeń. Zaletą takiego rozwiązania jest wzrost szybkości poprzez szybsze przełączanie kontekstu zadań oraz brak narzutu czasowego związanego z przechodzeniem procesora w inny tryb uprzywilejowania. Mechanizm ten umożliwia także bezpośrednią komunikację pomiędzy zadaniami RT poprzez współdzielenie danych. Wadą jest natomiast brak ochrony pamięci co może mieć wpływ na niezawodność systemu.

Szeregowanie zadań w RTLinux'ie

Scheduler RTLinux'a, działający jako odrębny moduł, używa algorytmu szeregowania *Priority-Based Rate Monotonic Scheduling Algorithm* (RMS). Algorytm ten bazuje na stałych priorytetach zadań wybierając zawsze do wykonania zadanie gotowe o najwyższym priorytecie. Zadaniom nie przydziela się kwantu czasu, tak jak dzieje się to w standardowym schedulerze Linuxa – mogą one być wywłaszczone tylko przez zadanie RT o wyższym priorytecie lub same oddać procesor. Zwykłe jądra Linux'a, działające jako jedno z zadań RT, ma najniższy priorytet (jest procesem *idle)* więc wykonywane jest tylko wtedy gdy żadne z pozostałych zadań RT nie ubiega się o procesor. W przeciwieństwie do zwykłego Linux'a, RTLinux nie buduje listy zadań gotowych – wszystkie trzymane są na jednej liście (w przypadku SMP, jednej dla każdego procesora).

Schemat działania algorytmu szeregowania używanego w RTLinux'ie

```
rtl_schedule() {
  for (każdy proces realtime)
    wybierz proces o najwyższym priorytecie;
  if (wybrano proces) {
    for (każdy wstrzymany proces o wyższym priorytecie niż proces wybrany)
      wybierz proces, który powinien być wznowiony najwcześniej;
    if (wybrano wstrzymany proces)
      ustaw zegar na czas budzenia procesu;
  }
  wznów wybrany proces;
}
```

Odmierzanie czasu

W RTLinux'ie zamiast generowania przerwań zegarowych generowane są przerwania po upływie zadanego czasu (*time-out*) umożliwia to uzyskanie rozdzielczości czasu na poziomie jednej mikrosekundy. Realizowane jest to poprzez programowanie kontrolera Intel 8254. Sposób ten zapewnia zmniejszenie kosztów stałych związanych z obsługą przerwań zegarowych. RTLinux udostępnia dwa tryby działania zegara: okresowy (*RTL_CLOCK_MODE_PERIODIC*) i „na żądanie" (*RTL_CLOCK_MODE_ONESHOT*). Okresowe przerwania o częstotliwości 100 Hz są emulowane dla zwykłego jądra Linux'a. Realizowane jest to poprzez mechanizm przerwań oczekujących.

Komunikacja międzyprocesowa

Operacje typu zapis na dysk, wizualizacja danych wykonywane są w RTLinux'ie poprzez zwykłe procesy wykonywane w trybie użytkownika, dlatego też niezbędne jest wprowadzenie mechanizmów do komunikacji pomiędzy zadaniami RT a zwykłymi procesami.

Niemożliwe jest użycie mechanizmów komunikacji udostępnianych przez standardowe jądro, gdyż może ono być wywłaszczone w dowolnym momencie. RTLinux sam dostarcza kilku mechanizmów komunikacji. Najważniejsze z nich to: kolejki czasu rzeczywistego (*RT-FIFOS)* oraz pamięć dzielona. Do synchronizacji zadań RT używane są semafory i muteksy.

Kolejki czasu rzeczywistego (*RT-FIFO)* – są to bufory alokowane w przestrzeni jądra. Mogą one być odczytywane i zapisywane zarówno przez zwykłe procesy Linux'a jak i przez zadania RT. Kolejki *RT-FIFO* są jednokierunkowe czyli do uzyskania dwukierunkowego połączenia należy użyć dwóch kolejek. Kolejki istnieją jako urządzenia znakowe w katalogu */dev* (*/dev/rtf0, /dev/rtf1, /dev/rtf2, itd.).* Są one obsługiwane przez osobny moduł RTLinux'a.

Pamięć dzielona – może służyć do wymiany większych ilości danych pomiędzy zadaniami RT i zwykłymi procesami Linux'a. Jest to też najszybszy sposób komunikacji pomiędzy zadaniami RT, ze względu na brak zaangażowania systemu w realizację tej techniki.