Systemy czasu rzeczywistego

Co to jest system czasu rzeczywistego RTS (*Real-Time System*)?

Często proponowana definicja RTS mówi, że jest to system, w którym można określić maksymalny czas wykonania poszczególnych operacji.

Szersza definicja: w systemie RTS poprawność procesu obliczeniowego zależy nie tylko od samego wyniku, ale również od czasu, w którym został on osiągnięty.

Oczywiste:

- sterowanie produkcją
- aparatura medyczna
- sterowanie elektroniką samochodową, np. ABS
- lotnictwo i aeronautyka

Mniej oczywiste:

- systemy nawigacji
- systemy rozpoznawania głosu
- odtwarzacze multimedialne, np. MP3
- systemy telekomunikacyjne
- urządzenia konsumenckie

Czy/kiedy potrzebny jest nam system operacyjny?

W projektowaniu systemów czasu rzeczywistego i systemów wbudowanych możliwe są dwa podejścia:

- bez systemu operacyjnego — aplikacja sama zarządza zasobami komputera

  Rozwiązanie stosowane w przypadku mniejszych aplikacji, w których programista jest w stanie zaimplementować wszystkie niezbędne mechanizmy: obsługi wątków, alokacji pamięci i zasobów, itp. Ponieważ aplikacja realizuje część funkcji systemu operacyjnego, to podejście nazywane jest czasem architekturą **pseudojądra**.
  Ma to szczególny sens w przypadku użycia procesora mniejszej mocy, z wolniejszym zegarem, z mniejszą ilością pamięci RAM.

- z systemem operacyjnym — aplikacja tworzy wątki, które działają pod kontrolą systemu operacyjnego, który zarządza pracą całego systemu i dostarcza wątkom usług, takich jak usługi szeregowania, przydziału pamięci, komunikacji i synchronizacji, podobnie jak w zwykłych systemach komputerowych

  Ponieważ system operacyjny generuje pewne narzuty, aplikację opłaca się zbudować w oparciu o system operacyjny jeśli przekracza pewien stopień złożoności. Godzimy się wtedy na te narzuty w zamian za oferowane przez system usługi. Użycie systemu operacyjnego wiąże się często z koniecznością zastosowania mocniejszego systemu.

Ograniczenia czasowe RTS

Jakie konkretnie są te ograniczenia czasowe, których dotrzymanie powinien zagwarantować RTS?

- 1 sekunda?
- 1 milisekunda (10^{-3} sekundy)?
- 1 mikrosekunda (10^{-6} sekundy)?

Odpowiedź: każde z powyższych może być wymaganym czasem reakcji RTS.

Czy w dzisiejszych czasach wielordzeniowych procesorów z 4-gigahercowym (10^9 Hz) zegarem, zwykły system operacyjny nie mógłby pełnić roli RTOS?

Zwłaszcza gdyby nie był nadmiernie obciążony?
Czy nie będzie dostatecznie szybki?

Odpowiedź: być może, ale niekoniecznie. Jeśli jest choć niewielkie prawdopodobieństwo, że jakieś operacje systemowe nie zmieszczą się w określonych rygorach czasowych, to w warunkach rzeczywistej pracy, prędzej czy później taka sytuacja wystąpi, i taki system po prostu nie spełni wymagań czasowych aplikacji, aczkolwiek tylko z rzadka.

Kiedy RTS jest naprawdę potrzebny?

Rozważmy dalej możliwość stosowania 4-GHz wielordzeniowych procesorów z zapasem mocy, do prostych zadań, w miejsce RTS. Czy sporadyczne niedotrzymanie rygorów czasowych operacji ma wielkie znaczenie praktyczne?

Odpowiedź: to zależy od aplikacji i od wymagań użytkownika. Łatwo wyobrazić sobie, że system sterowania silnikiem odrzutowym musi zareagować w odpowiednim czasie na sytuację wymagającą natychmiastowego odcięcia dopływu paliwa.

Dla odmiany: system sterowania reklamą świetlną może być akceptowalny nawet wtedy, gdy będzie okresowo na chwilę zatrzymywał przesuwanie napisu na wyświetlaczu. Albo gdy system telefonii VOIP w pewnych sytuacjach opóźni zakodowanie i wysłanie co któregoś pakietu dźwiękowego.

Ale uwaga: aplikacja dekodująca obraz w odtwarzaczu multimedialnym może być równie bezużyteczna jak zły system sterowania silnikiem, jeśli będzie systematycznie spóźniała się ze zdekodowaniem na czas co którejś klatki obrazu wideo.

Kiedy RTS jest naprawdę potrzebny (cd.)?

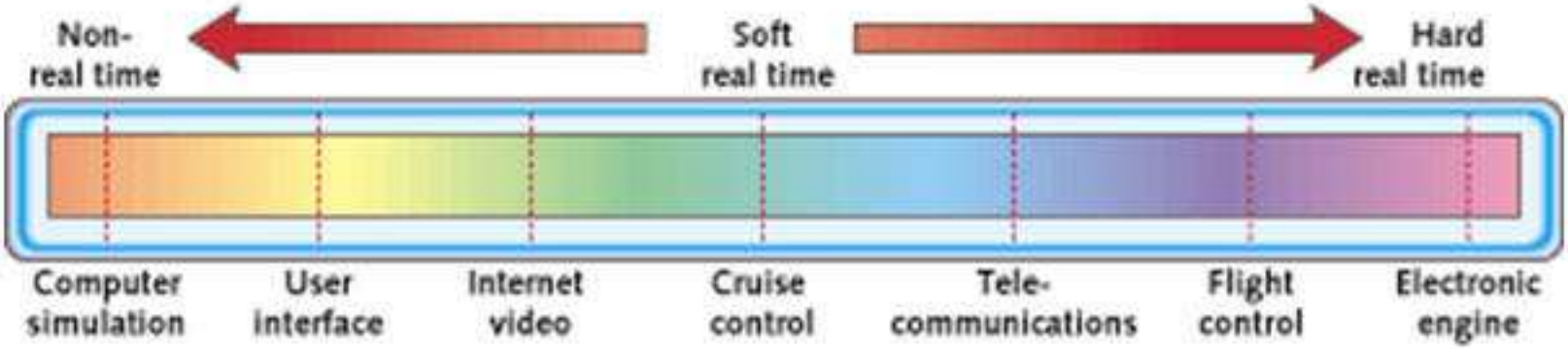
Odpowiedź 2: Jednak czasami można osiągnąć zaplanowany cel w systemie, który gwarantuje dotrzymanie danych reżimów czasowych na ogół, ale nie zawsze.

Silny system czasu rzeczywistego (*hard RTS*) musi bezwzględnie spełniać wszystkie wymagania RTS, i nadaje się do zastosowań, w których ich niedotrzymanie może powodować niebezpieczeństwo, katastrofę, itp.

Przykład: sterowanie lotem

Słaby system czasu rzeczywistego (*soft RTS*) dopuszcza pewne odchyłki od wyznaczonych reżimów czasowych, aczkolwiek ogólnie musi również działać szybko i niezawodnie. Systemy takie nadają się do zastosowań, w których nieznacznie spóźniona reakcja nadal może być przydatna.

Przykład: rozrusznik serca



Kiedy RTS jest naprawdę potrzebny (cd.)?

Odpowiedź 3: Wydajne procesory o dużej mocy obliczeniowej są kosztowne i mają duże zapotrzebowanie na moc zasilania. Jeśli zadanie nie wymaga dużej mocy obliczeniowej, to można je czasem zrealizować za pomocą systemu z 200-MHz mikrokontrolerem o minimalnych wymaganiach.

Wymaga to jednak zastosowania w RTS innego podejścia — pewnego minimalizmu, prostoty konstrukcji, i konsekwencji w projekcie całej architektury systemu. Ułatwia to zaprojektowanie systemu wykonującego dokładnie zaplanowane zadanie i nic więcej, a potem dokonanie jego analizy, w celu obliczenia najdłuższych ścieżek obliczeń, maksymalnych opóźnień, zlokalizowania wąskich gardeł, itp.

Wymagania systemów czasu rzeczywistego

Dla spełnienia wymagań czasu rzeczywistego jego projektant musi określić:

- kiedy pewne operacje muszą być wykonane
- kiedy pewne operacje muszą być zakończone
- co zrobić w sytuacji, gdy wszystkie wymagania czasowe nie mogą być jednocześnie spełnione

Systemy czasu rzeczywistego nie są podobne do „zwykłych" systemów komputerowych. Są inaczej projektowane i inaczej wykorzystywane. Zawierają inne podsystemy. Dostarczają programiście innych narzędzi. Przerzucają na programistę zadanie globalnego sterowania przydziałem zadań i priorytetów.

Dodatkowymi wymaganiami, często stawianymi systemom czasu rzeczywistego, jest praca z niskim poborem mocy zasilania, często związana z zasilaniem bateryjnym, oraz wysoka niezawodność. Te cechy wiążą systemy czasu rzeczywistego z **systemami wbudowanymi** (*embedded systems*).

Krótkie podsumowanie — pytania sprawdzające

1. Co to jest system czasu rzeczywistego?

2. Kiedy system czasu rzeczywistego powinien być budowany na bazie systemu operacyjnego czasu rzeczywistego?

3. Czym różni się silny system czasu rzeczywistego od słabego?

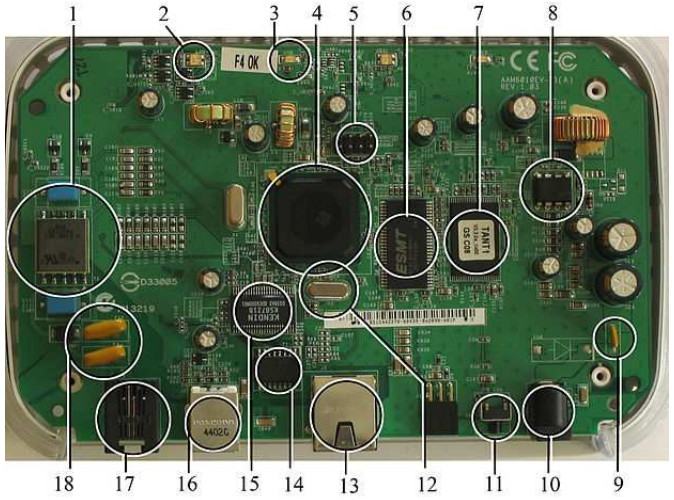
4. Jakie są podstawowe wymagania systemów czasu rzeczywistego?

Systemy wbudowane

"**System wbudowany** to system komputerowy z dedykowaną funkcją w ramach większego systemu mechanicznego lub elektrycznego, często z ograniczeniami obliczeniowymi czasu rzeczywistego. Jest wbudowany jako część kompletnego urządzenia, często zawierającego części sprzętowe i mechaniczne. Systemy wbudowane kontrolują obecnie wiele powszechnie używanych urządzeń. Dziewięćdziesiąt osiem procent wszystkich mikroprocesorów produkowanych jest jako komponenty systemów wbudowanych.

Przykładami właściwości typowych komputerów wbudowanych w porównaniu z odpowiednikami ogólnego przeznaczenia są niskie zużycie energii, małe rozmiary, wytrzymałe zakresy pracy i niski koszt jednostkowy. Dzieje się tak za cenę ograniczonych zasobów przetwarzania, co znacznie utrudnia programowanie i interakcję."

https://en.wikipedia.org/wiki/Embedded_system



Płytka modemu/routera ADSL: mikroprocesor (4), RAM (6), pamięć flash (7).


ATM

Systemy „bezgłowe”

Wiele (większość) systemów wbudowanych nie posiada monitora do komunikacji z użytkownikiem, jak również klawiatury, myszy, itp. Takie systemy określane są mianem *headless* (bezgłowe?). Ich istnienie jest wyrazem zarówno zastosowań do pracy całkowicie automatycznej, bez interakcji z ludźmi, jak i zwiększonego kosztu budowy urządzeń, które posiadają te elementy interfejsu użytkownika.

Bezgłowe systemy wbudowane mogą posiadać czujniki takie jak: termometr, akcelerometr, żyroskop, i reagować na ich wskazania, i odpowiadają sterując urządzeniami wykonawczymi zwanymi **aktuatorami**, albo wyzwalając różne powiadomienia, alarmy, itp.

Systemy wbudowane „z głową”

Oczywiście istnieją również systemy wbudowane wyposażone w urządzenia interfejsu użytkownika: monitor(y), klawiaturę, itp. W miarę jak spada koszt sprzętu komputerowego, są one nawet coraz bardziej popularne (patrz np. artykuły gospodarstwa domowego z wyświetlaczem: pralki, kuchenki, piekarniki, zaparzarki do kawy, itp.). Pomimo iż są przystosowane do interakcji z ludźmi, często nie wyglądają jak typowe komputery.

Własności systemów wbudowanych

Systemy wbudowane bardzo różnią się między sobą, ale pewne charakterystyczne własności łączy wiele z nich:

- niski pobór mocy
- niewielkie rozmiary
- niski koszt produkcji za sztukę
- ograniczona moc obliczeniowa i pamięć
- szerokie spektrum warunków pracy
- długi czas eksploatacji

Krótkie podsumowanie — pytania sprawdzające

1. Co to jest system wbudowany?

2. Jakie są typowe cechy systemów wbudowanych?

3. Co łączy systemy wbudowane z systemami czasu rzeczywistego?

Systemy operacyjne czasu rzeczywistego

W systemach operacyjnych czasu rzeczywistego RTOS (*Real Time Operating System*) nie ma zasady równości ani sprawiedliwości, która przyświeca zwykłym systemom operacyjnych GPOS (*General Purpose Operating System*). Na przykład, wątek o wysokim priorytecie może wykonywać się tak długo jak zechce, uniemożliwiając wykonanie wątków o niższym priorytecie (oczywiście o ile sam nie zostanie wywłaszczony przez wątek o jeszcze wyższym priorytecie).
Takie szeregowanie priorytetowe z wywłaszczaniem pozwala wątkom o wysokich priorytetach zmieścić się w wyznaczonym reżimie czasowym w RTOS.

System RTOS powinien nie tylko dostarczać mechanizmów i usług dla wykonywania, planowania, i zarządzania zasobami aplikacji, ale musi również sam sobą zarządzać w sposób oszczędny, przewidywalny, i niezawodny.

System operacyjny czasu rzeczywistego powinien być modularny i rozszerzalny.
W przypadku systemów wbudowanych, jądro systemu musi być małe, ze względu na lokalizację w ROM, i często ograniczoną wielkość pamięci RAM.

Podstawowymi zaletami jest prostota i oszczędność. RTOS może mieć mikrojądro dostarczające tylko podstawowych usług planowania, synchronizacji, i obsługi przerwań. Gdy niektóre aplikacje wymagają większego spektrum usług systemowych (systemu plików, systemu wejścia/wyjścia, dostępu do sieci, itp.), RTOS z mikrojądrem dostarcza tych usług w postaci zwykłych procesów.

Jądra, mikrojądra, i nanojądra

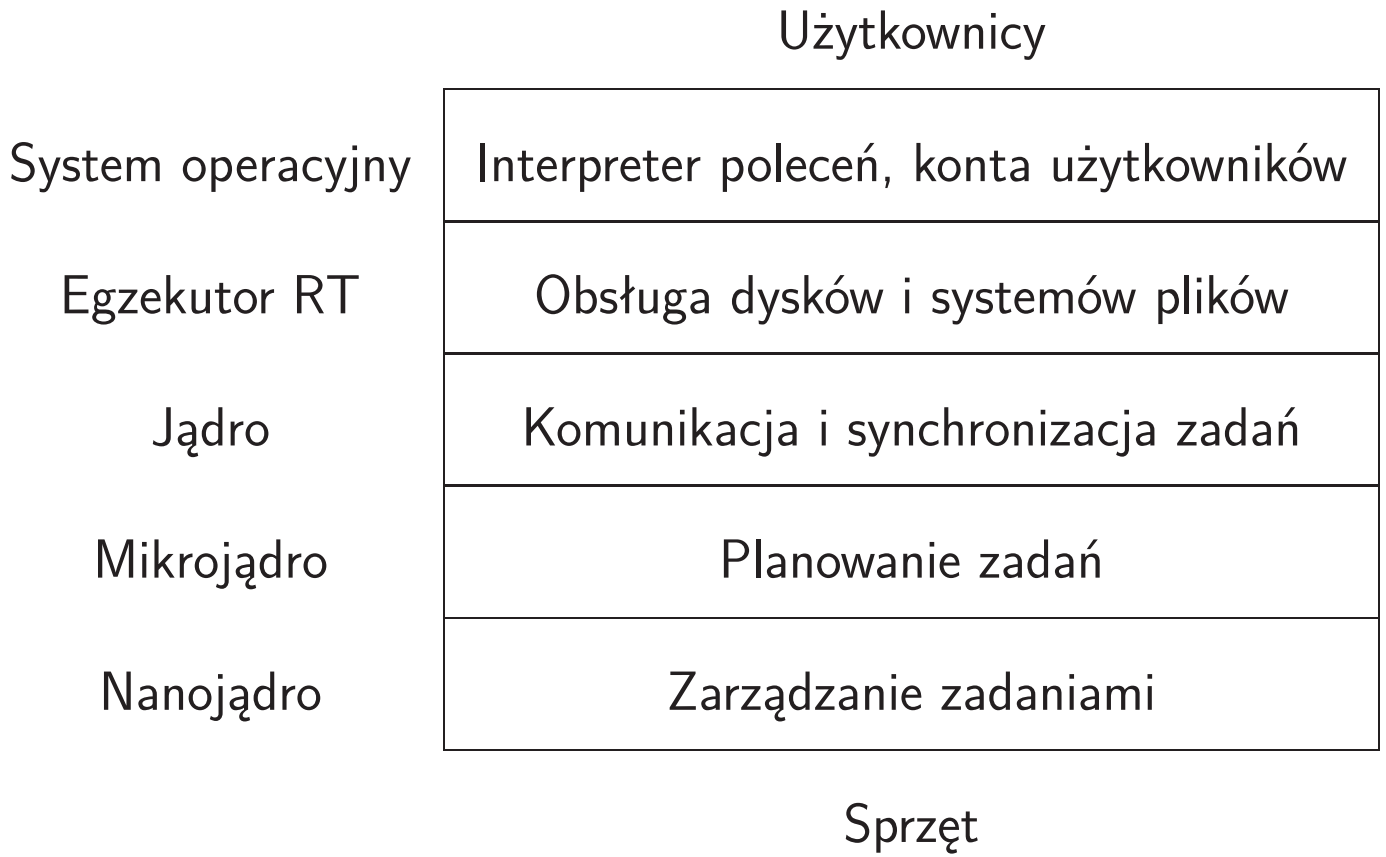
Jądro (*kernel*) systemu RTOS zawiera moduły realizujące co najmniej następujące trzy funkcje: **planowanie** (*scheduling*), **dyspozycja** (*dispatching*), i usługi komunikacji i synchronizacji.

Planista (*scheduler*) realizuje algorytm decydujący które zadanie ma być uruchomione w następnej kolejności w systemie wielozadaniowym. **Dyspozytor** (*dispatcher*), zwany również **ekspedytorem** administruje strukturami niezbędnymi do uruchamiania zadań. Mechanizmy komunikacji i synchronizacji obejmują: semafory, monitory, kolejki komunikatów, i inne.

Mikrojądrem nazywa się jądro o funkcjonalności ograniczonej do modułu planowania i dyspozytora. Inaczej mówiąc, mikrojądro jest w stanie realizować wielozadaniowość na poziomie procesów.

Nanojądro obsługuje jedynie zarządzanie wątkami.

Egzekutor RT i system operacyjny



Typowe komercyjne systemy RTOS są **egzekutorami RT** (*realtime executive*).

Krótkie podsumowanie — pytania sprawdzające

1. Jakie są przykładowe różnice pomiędzy zwykłymi systemami operacyjnymi GPOS a systemami operacyjnymi czasu rzeczywistego RTOS?

2. W jaki sposób system zrealizowany w architekturze mikro- lub nanojądra realizuje zadania normalnie realizowane w ramach jądra systemu operacyjnego?

Mechanizmy systemów operacyjnych czasu rzeczywistego

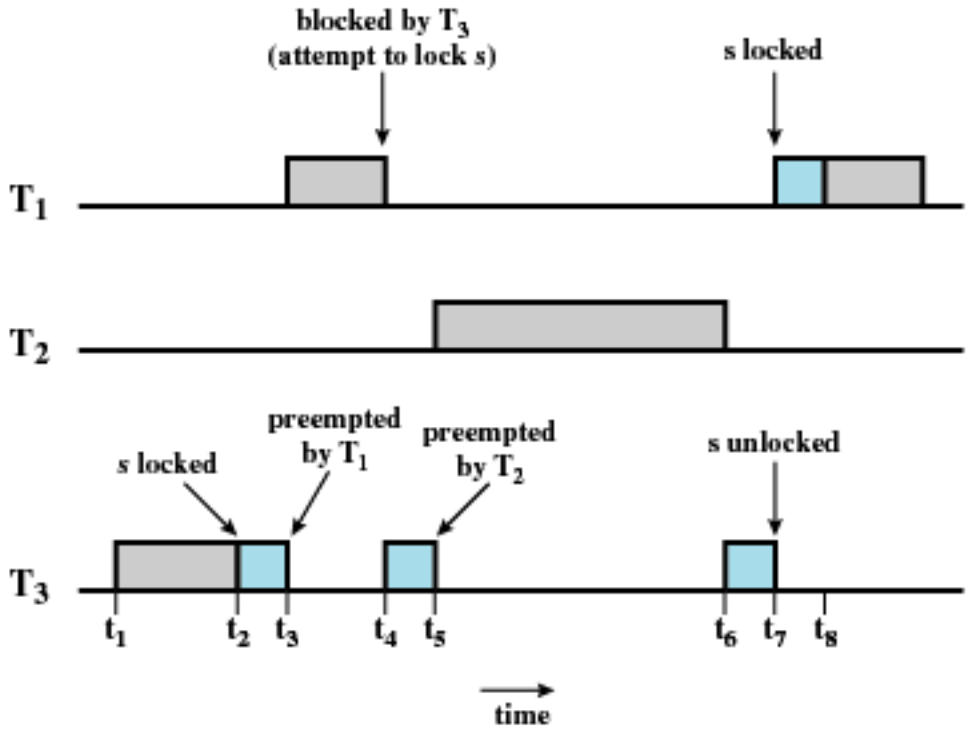
Mechanizmy systemu operacyjnego RTOS, które umożliwiają pracę aplikacji w czasie rzeczywistym to:

- przewidywalne szeregowanie wykonywania zadań
- mechanizmy zapobiegania inwersji priorytetów
- przewidywalne zarządzanie pamięcią
- przewidywalny maksymalny czas obsługi przerwań
- wywłaszczalne jądro systemu

Spośród tych zagadnień, mechanizmy szeregowania mające zastosowania do RTOS były omówione oddzielnie, w ramach wykładu o szeregowaniu zadań. Pozostałe mechanizmy zostaną tu pokrótce omówione.

Odwrócenie priorytetów

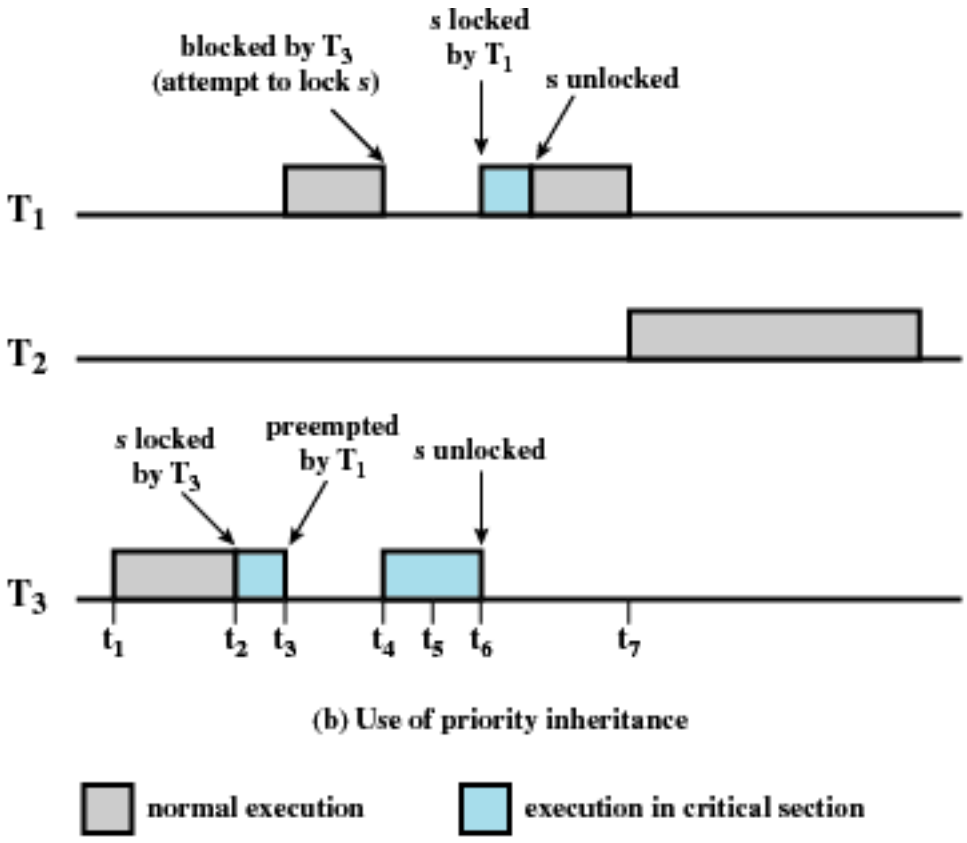
Jak widać na rysunku, zadanie o niskim priorytecie T_3 poprawnie uzyskuje dostęp do jakiegoś zasobu s (semafora), po czym zostaje wywłaszczone przez zadanie o wysokim priorytecie T_1. Jednak gdy T_1 zgłasza żądanie dostępu do s, wykonywanie wraca do T_3. Ta sytuacja, wynikająca z interakcji pomiędzy T_1 a T_3 może jednak nadal być poprawna — programista wiedział, że zadania T_1 i T_3 współdzielą zasób.



Problem powstaje dopiero, gdy zadanie T_1 zostanie zablokowane na czas dowolnie długi, gdyż T_3 może być wywłaszczone przez inne zadania, takie jak T_2, o priorytecie niższym niż T_1, prowadząc do **odwrócenia (inwersji) priorytetów**.

Odwrócenie priorytetów — dziedziczenie priorytetów

Metoda rozwiązania problemu odwrócenia priorytetów, zwana **dziedziczeniem priorytetów** (*priority inheritance*), polega na tym, że w momencie gdy zadanie o wyższym priorytecie T_1 zostaje zablokowane w oczekiwaniu na zasób s zajęty przez zadanie T_3 o niższym priorytecie, T_3 chwilowo dziedziczy wysoki priorytet T_1 i nie podlega wywłaszczeniu przez zadania typu T_2.



Odwrócenie priorytetów — pułap priorytetów

Inna metoda rozwiązania problemu inwersji priorytetów, zwana **pułapem priorytetów** (*priority ceiling*) polega na przypisaniu priorytetów zasobom, przy czym najniższy priorytet zasobu jest wyższy od najwyższego priorytetu wszystkich zadań. W chwili zajęcia zasobu zadanie uzyskuje chwilowo priorytet równy temu zasobowi, i może wykonywać się do chwili zwolnienia zasobu, kiedy jego priorytet jest obniżany do pierwotnego poziomu.

Odwrócenie priorytetów — misja Pathfindera

Misja sondy marsjańskiej Pathfinder wystrzelonej w 1996 roku jest znana z kilku powodów, z których jednym jest wyjątkowo krótki czas przygotowania i skromny budżet — 150 M$ — porównywalny z budżetem niektórych produkcji filmowych. Innym symbolem tej misji był łazik marsjański Sojourner, który wylądował na Marsie 4 lipca 1997, i wykonał wysokiej rozdzielczości zdjęcia powierzchni Marsa.

Jednak misja została zagrożona przez błąd w systemie sterującym lądownika, który co prawda został wcześniej zaobserwowany w testach poprzedzających start, lecz został zlekceważony, ponieważ nie miał związku z oprogramowaniem lądowania.

Po wylądowaniu Pathfinder rozpoczął gromadzenie danych meteorologicznych (między innymi). W trakcie tych operacji zaczęło dochodzić to pełnego restartu jego systemu, co powodowało utratę danych i opóźnienia.

W dniach 5-14 lipca 1997 system wykonał cztery restarty, i powstało zagrożenie dla kontynuacji właściwej pracy.

Pathfinder posiadał magistralę do przesyłania danych. Niezbyt często wykonywane zadanie gromadzenia danych meteorologicznych miało niski priorytet, lecz zajmowało magistralę do przesłania tych danych, poprawnie blokując mutex. Najwyższy priorytet miało zadanie obsługi magistrali, lecz okresowo musiało krótko czekać na zwolnienie magistrali przez zadanie meteorologiczne. W tym krótkim oknie czasowym mogło wystąpić przerwanie wywołujące zadanie komunikacyjne średniego priorytetu, które nie zajmowało magistrali.

Jest to zatem klasyczny scenariusz inwersji priorytetów.

Nadmiernie przedłużający się czas nieaktywności zadania wysokiego priorytetu spowodowało przeterminowanie timera watchdoga, który zainicjował pełny restart systemu. Zaplanowane zadania zostały wtedy przesunięte na następny dzień.

Komputer na pokładzie lądownika był oparty na procesorze IBM Risc 6000 Single Chip (wersja uodporniona na promieniowanie kosmiczne), z 128MB RAM, 6MB EEPROM, z systemem operacyjnym VxWorks. System ten posiada mechanizm dziedziczenia priorytetów, lecz domyślna konfiguracja systemu ma ten mechanizm wyłączony.

Problem został zdiagnozowany przez intensywne testy na dokładnej kopii systemu łazika, które pozwoliły zduplikować zjawisko. 21 lipca 1997 udało się wgrać poprawkę włączającą dziedziczenie priorytetów, i misja mogła być kontynuowana.

Zarządzanie pamięcią

W nowoczesnych i większych systemach operacyjnych wiele funkcji zarządzania pamięcią jest obsługiwanych przez system, ze wsparciem od strony sprzętu, pozostawiając procesom swobodę wykorzystania pamięci. Jednak te usługi wiążą się z nietrywialnymi i nieprzewidywalnymi czasami wykonania, i korzyści wynikające z ich wykorzystania mogą być okupione zagrożeniami dla spełnienia wymagań czasowych przez system czasu rzeczywistego.

Przyjrzymy się więc tym mechanizmom, i stosowanym w systemach czasu rzeczywistego rozwiązaniom alternatywnym.

Funkcje zarządzania pamięcią:

- alokacja i dealokacja (przydział i zwalnianie) pamięci przez proces, i w tym celu: stronicowanie, wymiatanie i przywracanie procesów,
- obsługa pamięci wirtualnej, translacja adresów wirtualnych,
- ochrona pamięci — zapewnienie by programy odwoływały się jedynie do swoich własnych obszarów pamięci.

Zarządzaniem pamięcią zajmuje się jądro systemu, z wykorzystaniem sprzętowej jednostki zarządzania pamięcią MMU (*Memory Management Unit*).

Alokacja i dealokacja pamięci

Podstawową usługą dotyczącą pamięci dostarczaną przez systemy operacyjne, w tym również systemy RTOS, jest **alokacja** i **dealokacja** pamięci. Program żądający w trakcie pracy dynamicznego przydziału dodatkowych bloków pamięci dostaje je od systemu, a po wykorzystaniu zwalnia je, pozwalając systemowi ponownie wykorzystać je do innych celów.

Najprostszymi metodami alokacji pamięci są metody **alokacji ciągłej**, przydzielające ciągłe bloki pamięci. Ich wadą jest **fragmentacja** pamięci. Po przydzieleniu pewnej liczby bloków o różnej wielkości, uzyskanie kolejnego większego bloku może nie być możliwe, w sytuacji kiedy mniejsze bloki są nadal wolne. Problem fragmentacji normalnie rozwiązuje się przez **defragmentację**, która jednak w systemach czasu rzeczywistego jest trudna, ponieważ zarówno konieczność jej wykonania jak i potrzebny nakład czasu są trudne do przewidzenia.

Fragmentacja nie występuje w systemach **alokacji stronicowanej**, które dzielą pamięć fizyczną na niewielkie bloki zwane ramkami, z jednoczesnym podziałem przestrzeni adresowej procesu na identycznego rozmiaru bloki zwane stronami. Każdy proces posiada tablicę stron, i każde odwołanie do pamięci podlega translacji adresu z wykorzystaniem tej tablicy.

Alokacja i dealokacja pamięci — wnioski

Ciągła alokacja pamięci przez system, połączona z okresową defragmentacją jest nie do przyjęcia w systemach czasu rzeczywistego.

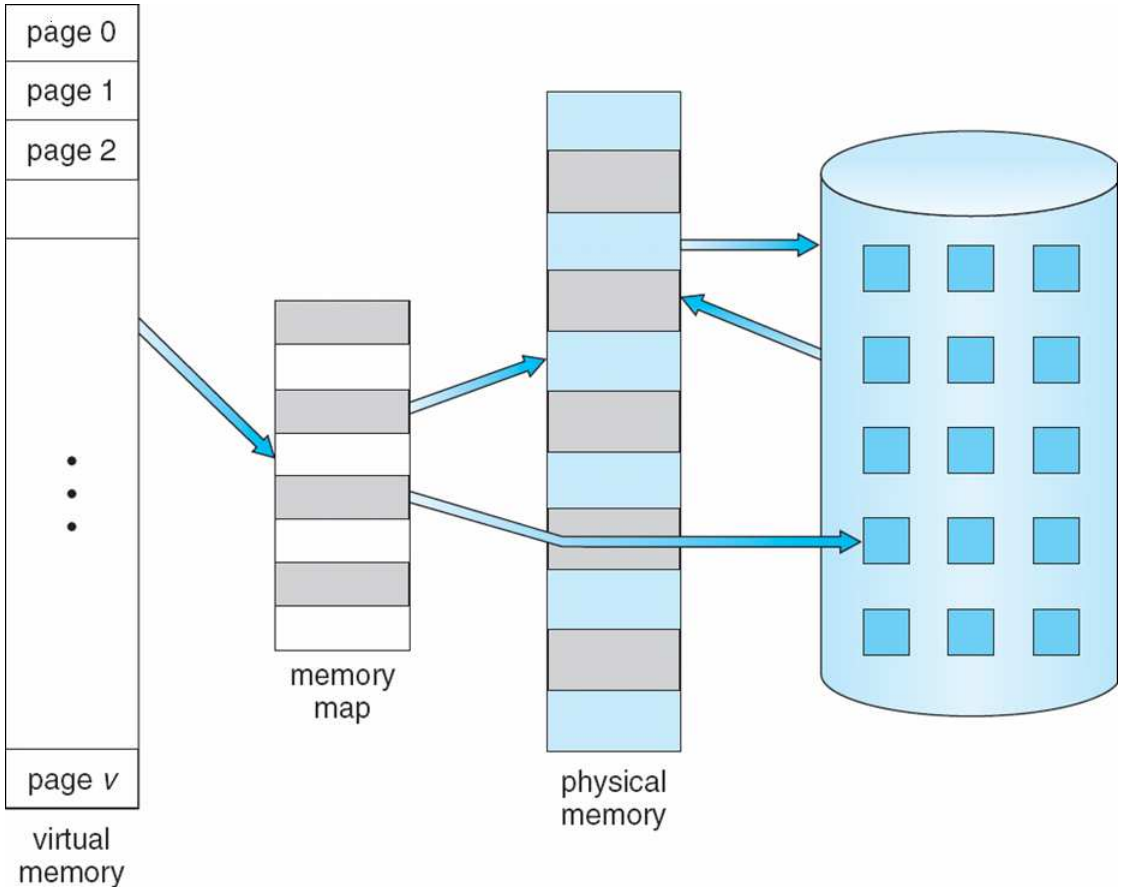
W najprostszym przypadku można założyć, ze system będzie budowany w taki sposób, że w całości będzie rezydował w pamięci RAM, i w czasie pracy nie będzie żądał dodatkowej alokacji pamięci. Zadanie nie zostanie utworzone, jeśli system nie zdoła zaalokować pamięci wystarczającej dla szczytowych wymagań tego zadania.

W przypadku alokacji stronicowanej, jest teoretycznie dopuszczalne stosowanie dynamicznej alokacji i dealokacji pamięci dla systemu czasu rzeczywistego. Jednak zwykłe mechanizmy dynamicznego przydziału pamięci (malloc/free) są typowo dość wolne, w porównaniu np. z samodzielnym zarządzaniem pamięcią przez wątek, który wstępnie przydzielił sobie pewną pulę pamięci.

Zatem nawet w systemie alokacji stronicowanej, zadania czasu rzeczywistego mogą stosować metodą wstępnego przydziału RAM, i nie korzystać z systemowych mechanizmów pamięci dynamicznej.

Pamięć wirtualna

Najczęściej wykorzystywanym w systemach operacyjnych schematem zarządzania pamięcią jest mechanizm **pamięci wirtualnej**. Polega on na całkowitym odseparowaniu pamięci logicznej użytkownika od pamięci fizycznej obsługiwanej przez system. Program użytkownika posługuje się wyłącznie liniową, adresowaną w sposób ciągły od zera pamięcią, natomiast system dzieli tę pamięć na małe fragmenty zwane stronami, i sam decyduje, które z nich są w danej chwili **rezydentne** w pamięci fizycznej. Oprócz tego wszystkie strony zapisane są na dysku w specjalnym **obszarze wymiany** (*swap space*).



Zalety i wady pamięci wirtualnej

W schemacie pamięci wirtualnej możliwe jest wykonywanie procesu, którego tylko niewielka część — w postaci wybranych stron pamięci — jest załadowana do pamięci fizycznej. Ponadto, jest możliwe współdzielenie stron pamięci przez różne procesy, co ma zastosowanie np. przy korzystaniu z bibliotek współdzielonych.

W celu realizacji pamięci wirtualnej potrzebne są jednak odpowiednie mechanizmy, których część wymaga realizacji lub wsparcia sprzętowego:

- **stronicowanie**, czyli sprowadzanie potrzebnych stron z dysku do pamięci,
- **zastępowanie stron**, czyli usuwanie z pamięci chwilowo niepotrzebnych stron.

Zasadniczą wadą tego mechanizmu z punktu widzenia systemów czasu rzeczywistego jest nieprzewidywalność czasu dostępu do pamięci w przypadku błędu strony. Gdy wątek o krytycznych wymaganiach czasowych zostanie uruchomiony, może się okazać, że jakaś strona jego pamięci nie znajdzie się w RAM. Dodatkowo mechanizm wymiatania całych zadań w przypadku całkowitego wyczerpania pamięci pozostaje w całkowitej sprzeczności z wymaganiami systemów czasu rzeczywistego.

Z tych względów, systemy operacyjne czasu rzeczywistego często nie zapewniają mechanizmów obsługi pamięci wirtualnej: odwzorowania wirtualnej przestrzeni adresowej dla procesów, stronicowania na żądanie, zastępowania stron, ani wymiatania.

Blokowanie pamięci

Są jednak powody, dla których system operacyjny RTOS może stosować pamięć wirtualną. Na przykład, może to być wymagane dla wsparcia narzędzi do tworzenia oprogramowania, jak edytory, debuggery, profilery, które typowo mają duże wymagania pamięciowe, choć nie wymagają pracy w czasie rzeczywistym. W takim przypadku, dla zapewnienia współpracy aplikacji czasu rzeczywistego z takimi aplikacjami czasu podzielonego, system RTOS musi dostarczać mechanizmów pozwalających na kontrolowanie stronicowania.

Specyfikacja POSIX dostarcza mechanizmu **blokowania** pamięci. Zadanie może zablokować w pamięci fizycznej zarówno cały swój kod, jak i pewien zakres adresowy (co wymaga dobrej znajomości rozlokowania aplikacji w pamięci). W ten sposób zadanie czasu rzeczywistego może pracować w systemie pamięci wirtualnej, ale jego pamięć nie będzie podlegać usuwaniu z RAM.

Zabezpieczenie pamięci

Inną podstawową usługą zapewnianą przez systemy operacyjne ogólnego przeznaczenia jest **zabezpieczenie** obszarów pamięci. Oznacza to zabezpieczenie pamięci systemu przed zadaniami, zadań przed innymi zadaniami, oraz zadań przed samymi sobą. Realizacja tych mechanizmów zabezpieczenia jest kosztowna, ponieważ wymaga specjalnej, kontrolowanej realizacji każdego dostępu do pamięci.

Wiele systemów RTOS nie zapewnia mechanizmów zabezpieczenia pamięci dla zadań jądra ani aplikacji użytkownika. Jak zwykle, argumentem za stosowaniem pojedynczej przestrzeni adresowej jest prostota (z punktu widzenia systemu) i mniejsze narzuty tego rozwiązania.

Niestety, te zalety okupione są większą komplikacją projektu aplikacji użytkownika, zwłaszcza, gdy trzeba wprowadzać modyfikacje. Zatem to rozwiązanie jest właściwe jedynie dla bardzo niewielkich systemów wbudowanych. Natomiast wiele systemów RTOS zapewnia ochronę pamięci procesów.

Pewną alternatywą jest możliwość konfiguracji mechanizmów pamięci wirtualnej oferowana przez niektóre systemy RTOS.

Obsługa przerwań zewnętrznych

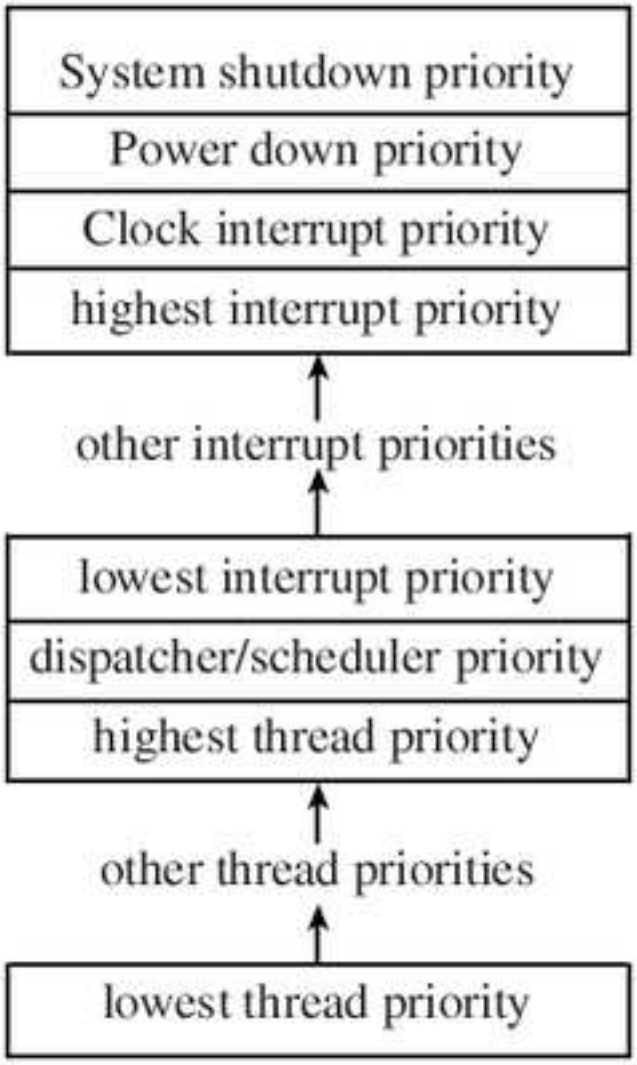
Przerwania sprzętowe stanowią mechanizm powiadamiania aplikacji o zdarzeniach zewnętrznych i obsługi sporadycznych operacji I/O. Nakład czasu niezbędny do obsługi przerwania zależy od jego źródła.

W szczególności, obsługa przerwań DMA wymaga znacznego czasu. Na przykład, dla obsługi nadchodzącego pakietu sieciowego, interfejs sieciowy generuje przerwanie. Dla jego obsługi jądro musi wywołać funkcję obsługi odpowiedniego protokołu. Ta z kolei identyfikuje zadanie, które ma otrzymać pakiet, kopiuje go do bufora w przestrzeni adresowej zadania, wykonuje dodatkowe czynności wymagane przez protokół (np. generuje komunikat potwierdzenia), itp. Wymagany na to czas może być długi i trudny do określenia.

Procedury obsługi przerwań generowanych przez system dyskowy i sieciowy typowo trwają setki mikrosekund do dziesiątek milisekund. Dlatego większość systemów dzieli ich obsługę na dwie części: natychmiastową i planowaną. Jest to tzw. **podzielona obsługa przerwań** (*split interrupt handling*).

Natychmiastowa obsługa przerwań

Pierwsza faza obsługi przerwania, zwana **natychmiastową obsługą przerwania**, wykonywana jest z zastosowaniem właściwego priorytetu. Priorytety przerwań są ogólnie wyższe niż priorytety zadań, jak również wyższe niż priorytet planisty.



Planowana obsługa przerwań

Oprócz przypadków bardzo trywialnych funkcji, natychmiastowa obsługa przerwania nie kończy jego właściwej obsługi. Właściwa faza obsługi przerwania — **planowana obsługa przerwania** — powinna być wywłaszczalna, i wykonywana z priorytetem odpowiednim dla danego urządzenia. Na przykład, może być on równy priorytetowi zadania, które zainicjowało operacje na urządzeniu. Dlatego na diagramie zadań jądra, po wykonaniu natychmiastowej obsługi przerwania wywoływany jest planista, który umieszcza zadanie planowanej obsługi przerwania w kolejce zadań gotowych.

Wywłaszczalne jądro w systemach RT

Typowe jądro systemu GPOS nie podlega wywłaszczaniu, tzn. funkcje jądra — zarówno operacje systemowe, np. obsługa przerwań, jak i zwykłe funkcje wywołane przez programy użytkowników — wykonywane są bez ograniczeń. W systemach RTOS może to powodować sytuacje, kiedy wykonywanie funkcji dla wątku o niskim priorytecie mogłoby trwać przez czas nieokreślony, lub ogólnie długi. W oczywisty sposób stoi to w sprzeczności z wymaganiami RTOS. Dlatego systemy RTOS mogą mieć jakąś formę wywłaszczania jądra.

Jednak procedury jądra każdego systemu mają okna czasowe, kiedy nie może dojść do jego wywłaszczenia. W tych okienkach nie działają mechanizmy czasu rzeczywistego, i muszą one być minimalizowane.

Niektóre systemy zapewniają wewnątrz procedur jądra **punkty wywłaszczania** (*checkpoints*), w których procedura może być przerwana i jądro wywłaszczone. Jednak analizując taki system z punktu widzenia wymagań RT należy pamiętać o uwzględnieniu wszystkich elementów jądra, w tym kodu sterowników.

Krótkie podsumowanie — pytania sprawdzające

1. Na czym polega zjawisko odwrócenia priorytetów?

2. Jakie są podstawowe mechanizmy rozwiązywania tego problemu?

3. Jakie zasady zarządzania pamięcią różnią systemy czasu rzeczywistego, od systemów operacyjnych ogólnego przeznaczenia?

4. Dlaczego w systemach czasu rzeczywistego przydatny jest mechanizm wywłaszczania jądra systemu, i jakie konsekwencje to powoduje?

5. Na czym polega opóźniona obsługa przerwania, i czym obsługa natychmiastowa różni się od obsługi planowanej?