Systemy czasu rzeczywistego

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

Często proponowana definicja RTS mówi, że są to systemy, w których można określić maksymalny czas wykonania poszczególnych operacji.

Szersza definicja: w systemach 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

Ograniczenia czasowe RTS

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

- 1 sekunda?
- 1 milisekunda?
- 1 mikrosekunda?

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

Czy RTS po prostu musi być bardzo szybki?

Czy w dzisiejszych czasach wielordzeniowych procesorów z 4-gigahercowym 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.)?

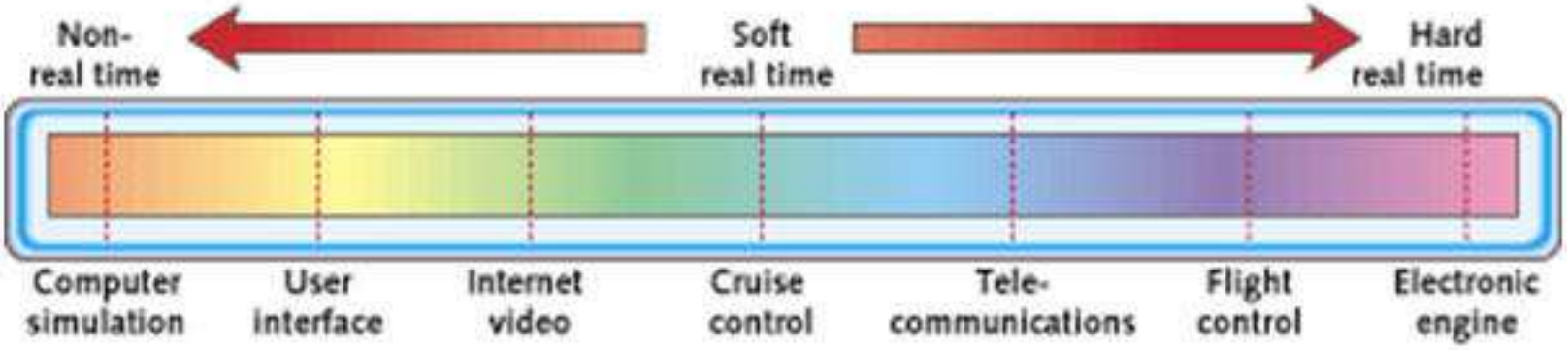
Odpowiedź 2: Jednak czasami można osiągnąć zaplanowany cel w systemie, który gwarantuje dotrzymanie danych reżimów czasowych na ogół, ale nie zawsze.

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 wbudowanego 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 operacyjnych. Są inaczej projektowane i inaczej wykorzystywane. Zawierają inne podsystemy. Dostarczają programiście innych narzędzi. Przerzucają na programistę zadanie globalnego sterowania przydziałem zadań i priorytetów.

Systemy operacyjne przeznaczonych do budowy systemów RTS, zwane systemami operacyjnymi czasu rzeczywistego (RTOS = *Real Time Operating System*) posiadają specjalne mechanizmy niezbędne przy uruchamianiu i eksploatacji RTS. Różnią się one zasadniczo od zwykłych systemów operacyjnych (GPOS = *General Purpose Operating System*)?

Systemy operacyjne czasu rzeczywistego

Na przykład, w systemach RT nie ma równości ani sprawiedliwości, którymi kierują się algorytmy szeregowania GPOS. 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 w RTOS zmieścić się w wyznaczonym reżimie czasowym.

System RTOS musi nie tylko dostarczać mechanizmów i usług dla wykonywania, planowania, i zarządzania zasobami aplikacji, ale powinien 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. Dlatego 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 zwykle dostarcza tych usług w postaci modułów dołączonych do jądra.

Czy/kiedy potrzebny jest nam system operacyjny?

W projektowaniu systemów czasu rzeczywistego i systemów wbudowanych rozważa się dwa możliwe podejścia:

- z systemem operacyjnym - aplikacja tworzy wątki, które działają pod kontrolą systemu operacyjnego, który zarządza pracą całego systemu i dostarcza usług wątkom, podobnie jak w zwykłych systemach komputerowych

- bez systemu operacyjnego - aplikacja jest jednym programem, który musi sam gospodarować zasobami komputera.

Rozwiązanie drugie stosuje się w przypadku mniejszych aplikacji, w których programista jest w stanie w miarę łatwo zaimplementować wszystkie niezbędne mechanizmy obsługi wątków, alokacji pamięci i zasobów, itp.

Jeśli aplikacja przekracza pewien stopień złożoności, to opłaca się ją zbudować w oparciu o system operacyjny, godząc się na pewne nieuchronne narzuty pobierane przez ten system, w zamian za oferowane przezeń usługi.

Krótkie podsumowanie — pytania sprawdzające

1. Co to jest system czasu rzeczywistego?

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

3. Jakie są podstawowe cechy systemów operacyjnych czasu rzeczywistego?

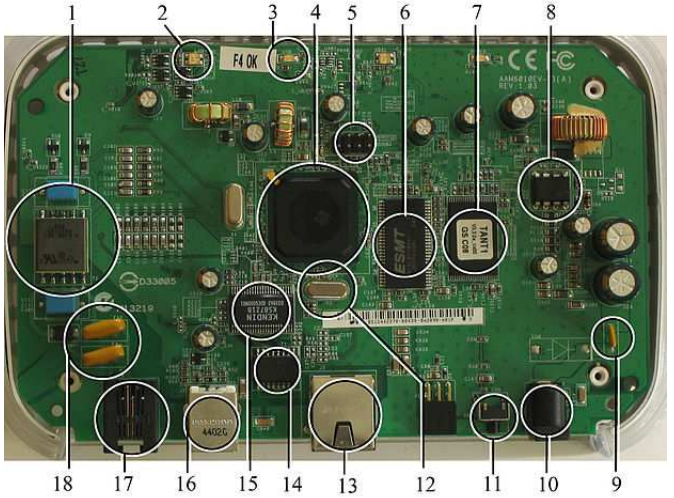
4. Kiedy system czasu rzeczywistego powinien być budowany na bazie systemu operacyjnego czasu rzeczywistego?

Systemy wbudowane

"**System wbudowany** to system komputerowy z dedykowaną funkcją w ramach większego systemu mechanicznego lub elektrycznego, często z ograniczeniami obliczeniowymi czasu rzeczywistym. 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 urzadzenia 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, nie wyglądają jak typowy komputer.

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. Co łączy systemy wbudowane z systemami czasu rzeczywistego?

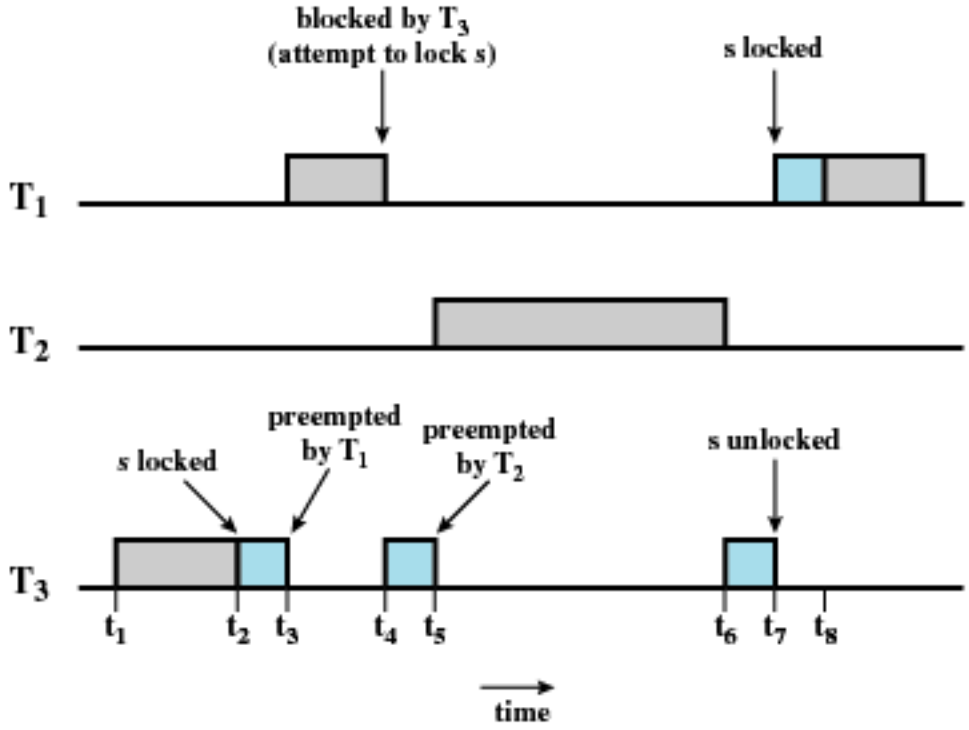
Mechanizmy systemów czasu rzeczywistego

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

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

Odwrócenie priorytetów

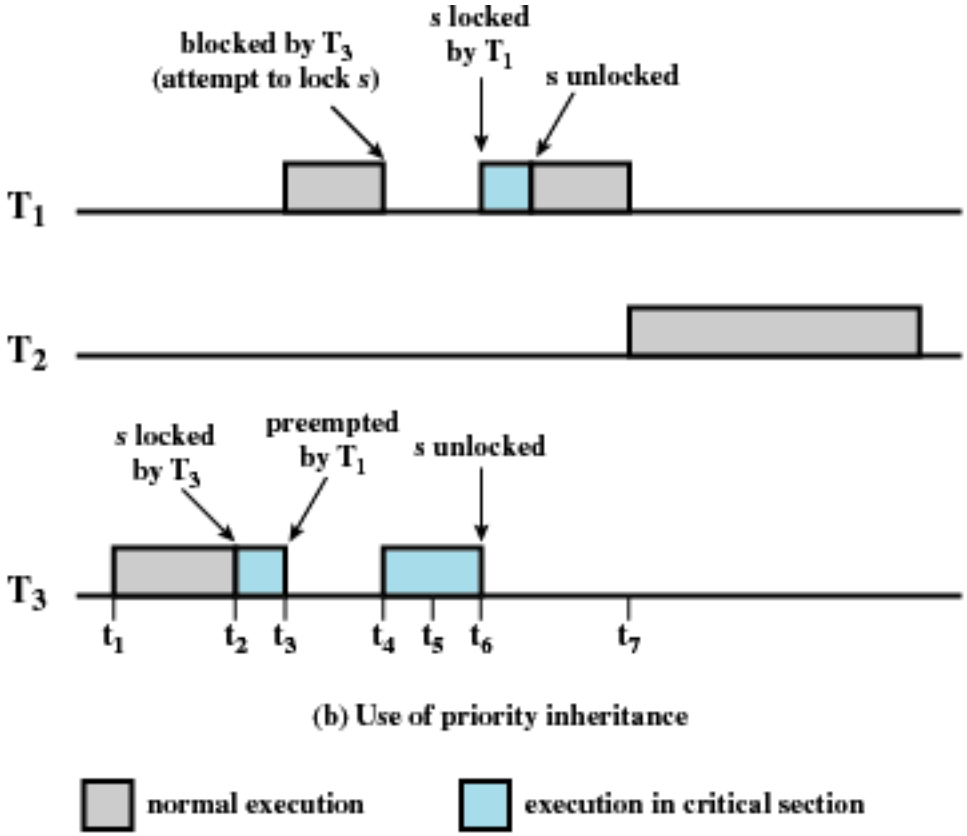
Jak widać na rysunku, zadanie o niskim priorytecie T_3 poprawnie uzyskuje dostęp do jakiegoś zasobu s (semafora), po czym zostaje wywłaszczone przez zadanie T_1 o wysokim priorytecie. Jednak gdy T_1 zgłasza żądanie dostępu do s, wykonywanie wraca do T_3. Ta sytuacja, wynikająca z interakcji pomiedzy 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 **dziedziczenia priorytetów** 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 wykonał wysokiej rozdzielczości zdjęcia powierzchni Marsa.

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

Problem powstawał w warunkach silnego obciążenia procesora zadaniami. Zadanie niskiego priorytetu zajęło potok do przesyłania danych, i poprawnie zablokowało odpowiedni mutex, lecz zostało następnie wywłaszczone przez zadanie średniego priorytetu. Z kolei zadanie wysokiego priorytetu chciało przesłać dane przez potok, lecz zostało zmuszone do oczekiwania na muteksie. Wkrótce potem manager zadań zauważył zadanie wysokiego priorytetu oczekujące przez zbyt długi czas, i wymusił restart systemu powodujący opóźnienie do następnego dnia.

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

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

Dopiero 21 lipca 1997 udało się wgrać poprawkę włączającą dziedziczenie priorytetów, i misja mogła być kontynuowana.

Krótkie podsumowanie — pytania sprawdzające

1. Na czym polega zjawisko odwrócenia priorytetów?

2. Jakie są podstawowe mechanizmy rozwiązywania tego problemu?

Zarządzanie pamięcią

Systemy czasu rzeczywistego często ignorują kwestie zarządzania pamięcią, ponieważ związane z nim algorytmy są często absorbujące i trudno przewidywalne.
W najprostszym przypadku można założyć, ze system będzie budowany w taki sposób, że w całości będzie rezydował w pamięci RAM. Zadanie nie zostanie utworzone, jeśli system nie zdoła zaalokować pamięci wystarczającej dla szczytowych wymagań tego zadania.

W szczególności, systemy operacyjne czasu rzeczywistego często nie zapewniają obsługi pamięci wirtualnej: odwzorowania wirtualnej przestrzeni adresowej dla procesów, stronicowania na żądanie, ani wymiatania. W zależności od konkretnych usług zarządzania pamięcią można podzielić systemy operacyjne na klasy.

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 wykorzystać je do innych celów.

Najprostszymi metodami alokacji pamięci są metody alokacji ciągłej, przydzielające jednolite bloki pamięci. Ich wadą jest fragmentacja pamięci. Po przydzieleniu pewnej liczby bloków o zmiennej wielkości, uzyskanie kolejnego większego bloku może nie być możliwe, w sytuacji kiedy mniejsze bloki są nadal wolne. Problem fragmentacji 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.

Alokacja stronicowana

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.

Jednak nieciągłość pamięci fizycznej i translacja adresów komplikuje operacje I/O wykonywane przez DMA. Operacje DMA są znacznie prostsze i wymagają mniejszych narzutów, jeśli wykorzystują ciągły obszar pamięci fizycznej. W przypadku nieciągłych obszarów pamięci, transfer DMA musi być podzielony na szereg transferów, z których każdy musi być indywidualnie zainicjowany przez procesor.

Najczęściej stosowanym systemem alokacji stronicowanej jest system pamięci wirtualnej. Zasadniczą wadą tego mechanizmu z punktu widzenia systemów czasu rzeczywistego jest nieprzewidywalność czasu dostępu do pamięci w przypadku błędu strony.

Blokowanie pamięci

Są jednak powody, dla których system operacyjny RTOS może stosować pamięć wirtualną. Na przykład, dla wsparcia narzędzi do tworzenia oprogramowania, jak edytory, debuggery, profilery, które typowo mają duże wymagania pamięciowe, choć nie wymagają pracy w czasie rzeczywistym. 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 zarówno zablokować cały swój kod w pamięci fizycznej, albo 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 pamięci RAM.

Zabezpieczenie pamięci

Inną podstawową usługą zapewnianą przez systemy operacyjne ogólnego przeznaczenia jest zabezpieczenie obszarów pamięci. Wiele systemów RTOS nie zapewnia mechanizmów zabezpieczenia pamięci dla zadań jądra ani aplikacja 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.

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.

Obsługa przerwań zewnętrznych

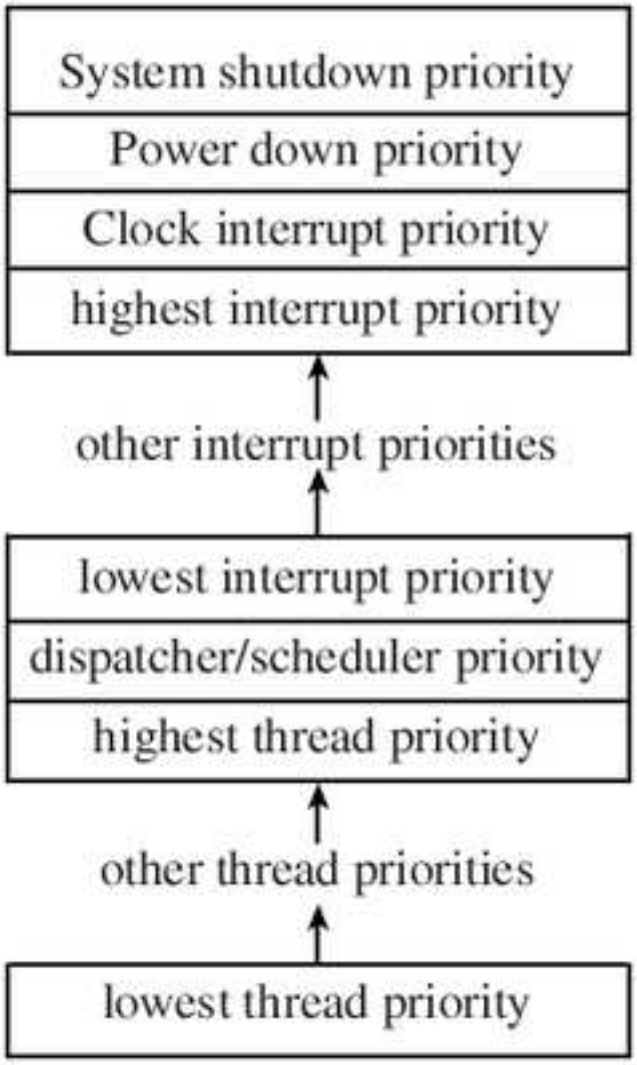
Przerwania sprzętowe stanowią mechanizm powiadamiania aplikacji o zdarzeniach zewnętrznych i obsługi sporadycznych operacji I/O. Nakład czasu niezbędny do obsługi przerwania zależy od jego źródła.

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.



W chwili przerwania, procesor wpisuje rejestry PC i stanu na stos przerwań, i wywołuje procedurę obsługi przerwania jądra. Jądro najpierw maskuje przerwania w celu bezpiecznego wykonania operacji na strukturach danych związanych z aktualnym przerwaniem. Następnie zapisuje stan procesora na stosie przerwań, odblokowuje przerwania, i dopiero wtedy wykonuje właściwy kod obsługi przerwania.

Więcej niż jedno urządzenie może być podpięte pod jedną linię przerwania. W takim przypadku wywoływane są po kolei procedury obsługi ich wszystkich. Procedura urządzenia, które nie wygenerowało przerwania wraca natychmiast.

Jeśli w czasie obsługi przerwania pojawia się inne przerwanie o wyższym priorytecie, to jest obsługiwane dokładnie w taki sam sposób.

Opóźnienie obsługi przerwania

Całkowity czas, jaki upływa od chwili wygenerowania przerwania, do momentu rozpoczęcia wykonywania jego obsługi, jest nazywany **opóźnieniem obsługi przerwania** (*interrupt latency*). Stanowi on miarę reaktywności systemu na zdarzenia zewnętrzne. Wielkość tego czasu jest sumą następujących elementów:

1. czas na dokończenie aktualnej instrukcji, obsługę stanu procesora, i zainicjowanie obsługi przerwania,
2. czas na zamaskowanie przerwań,
3. czas na natychmiastową obsługę przerwań o wyższym priorytecie, jeśli takie wystąpiły,
4. czas na wyskładowanie kontekstu wykonywanego zadania, i ustalenia które urządzenie zgłosiło przerwanie,
5. czas na wywołanie procedury obsługi.

Największą (i nieprzewidywalną) rolę odgrywa tu element 5.

Jedynym sposobem na zwiększenie reaktywności systemu jest zminimalizowanie całej procedury natychmiastowej obsługi przerwania.

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.

Krótkie podsumowanie — pytania sprawdzające

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

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

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