KaRTOS

8–bitowe jądro czasu rzeczywistego, część 1

Systemy czasu rzeczywistego są coraz częściej wykorzystywane przez konstruktorów urządzeń mikroprocesorowych. Po przełamaniu pierwszej bariery, którą jest zrozumienie mechanizmów działania RTOS i nieco inne konstruowanie algorytmów oprogramowania okazuje się, że stosując takie systemy można osiągnąć wiele korzyści.

Zaprezentowana w artykule „teoria systemów operacyjnych" jest mocno okrojona i ukierunkowana pod kątem autorskiego systemu KaRTOS dla małych mikrokontrolerów 8–bitowych. Zainteresowanych tym tematem Czytelników zachęcamy do zapoznania się z szerszymi opracowaniami dotyczącymi teorii wielozadaniowych systemów operacyjnych czasu rzeczywistego.

Nadeszła era mikrokontrolerów. Wniosek ten nasuwa się po niezbyt wnikliwej obserwacji otaczającego świata: pralki, drukarki, telefony, faksy, odtwarzacze CD, DVD, MP3, samochody, pociągi, samoloty, aparaty fotograficzne, skanery, tunery... jak widać zrobiła się całkiem spora lista. Wyparcie przez technikę mikroprocesorową tradycyjnej techniki cyfrowej operującej na bramkach i przerzutnikach pociągnęło za sobą zmianę charakteru pracy inżynierów i hobbystów elektroniki cyfrowej. Algebrę Boole'a z tablicami stanów zastąpiły algorytmy realizowane przez wspomniane układy programowalne. Rozwój mikroprocesorów (zwiększanie zasobów pamięci, mocy obliczeniowych, liczby urządzeń peryferyjnych) umożliwił z kolei rozbudowywanie i zwiększanie stopnia złożoności zadań kodowanych w programach komputerowych. Aby możliwe było zapanowanie nad relacjami i powiązaniami (logicznymi i czasowymi) panującymi pomiędzy jednostkami organizacyjnymi programów (funkcjami, blokami funkcjonalnymi) oraz uelastycznienie aplikacji jako całości, stosuje się system operacyjny – mówiąc w uproszczeniu: program do zarządzania pracą innych programów. Niniejsze opracowanie jest prezentacją wielozadaniowego mikrosystemu operacyjnego oraz kursem pisania aplikacji wykorzystujących jego zasoby. Czytelnicy mniej obeznani z tematyką systemów operacyjnych mogą potraktować materiał jako „miękkie", praktyczne wprowadzenie w ciekawy świat systemów operacyjnych.

Co to jest KaRTOS

KaRTOS jest edukacyjnym mikrosystemem operacyjnym czasu rzeczywistego dla popularnych mikrokontrolerów 8–bitowych RISC z rdzeniem AVR. Podczas projektowania systemu szczególny nacisk położono na prostotę, która gwarantuje niezawodność i szybkość działania. Modułowa budowa zapewnia elastyczność, a możliwości konfiguracyjne pozwalają na dostosowanie jego parametrów do aktualnych potrzeb aplikacji. KaRTOS został napisany w języku C z wykorzystaniem wstawek asemblerowych. Pisanie aplikacji korzystających z jądra KaRTOS jest bardzo proste i łatwe do opanowania w krótkim czasie.

Główne cechy systemu KaRTOS to:

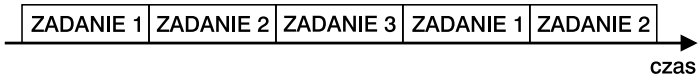
- wielozadaniowość – w systemie można uruchomić do 254 zadań, jeśli pozwalają na to zasoby sprzętowe,
- prostota, modułowość, łatwość konfigurowania,
- niewielki rozmiar pozwala na uruchomienie systemu na mikrokontrolerze posiadającym tylko 4 kB pamięci programu i 1 kB RAM–u,
- szeregowanie zadań zgodnie z wybranym algorytmem: prosty *round robin* lub priorytetowy algorytm karuzelowy z możliwością określenia maksymalnego czasu przydzielanego zadaniu, po którym zostanie ono wywłaszczone,
- zapewnienie komunikacji między zadaniami poprzez współdzielone obszary pamięci,
- synchronizacja z pomocą semaforów,
- energooszczędność uzyskiwana np. przez systemowy proces bezczynności wprowadzający mikrokontroler w tryb obniżonego poboru energii, gdy żadne zadanie aktualnie nie wykorzystuje jego mocy obliczeniowej,

Kilka definicji przydatnych do zrozumienia działania systemu KaRTOS

System operacyjny – System operacyjny to najprościej pisząc program do zarządzania pracą (współpracą) innych programów (aplikacji). Obserwując choćby działanie komputera PC zauważamy, że może on wykonywać kilka (kilkanaście) takich aplikacji jednocześnie, pomimo iż pracujący procesor w danej chwili czasowej jest w stanie wykonywać tylko jedną operację. Komputer najczęściej dysponuje ponadto tylko jedną stacją dyskietek, CD, DVD czy monitorem. Pomimo to wszystkie uruchomione programy mogą korzystać z wymienionych zasobów.

Zadanie – Zadanie jest samodzielnym programem wykonywanym przez procesor. Realizuje określony algorytm (funkcjonalność). Posiada własny kod i dane, na których wykonuje operacje. Może komunikować się z innymi zadaniami istniejącymi w systemie i korzystać z zasobów sprzętowych za pośrednictwem funkcji systemowych. Każde zadanie za sprawą systemu operacyjnego „czuje się" jakby samodzielnie wykorzystywało procesor. W związku z powyższym zadania są wykonywane asynchronicznie i współbieżnie. W przypadku pisania kodu przez kilku programistów każdy z nich może niezależnie od innych dysponować całym „wirtualnym kontrolerem" nie martwiąc się o ścisłe zależności czasowe występujące pomiędzy blokami funkcjonalnymi – każdy działa w obrębie swojego zadania (swoich zadań).

Wielozadaniowość – Wielozadaniowość jest cechą praktycznie wszystkich obecnych na rynku systemów operacyjnych. Objawia się ona możliwością wykonywania „jednocześnie" kilku (kilkunastu) zadań przez procesor z uruchomionym systemem wielozadaniowym.



Rys. 1. Podział czasu pracy kontrolera pomiędzy zadaniami

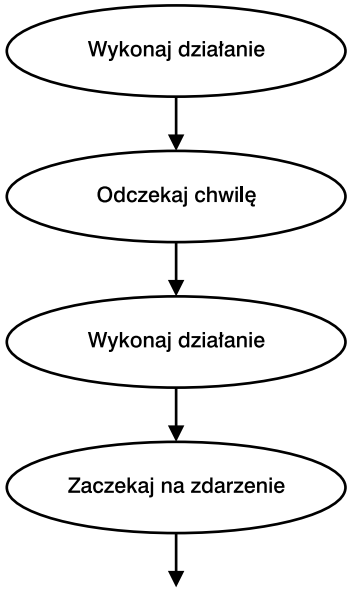
– odmierzanie czasu do roku 2255 dzięki wbudowanemu zegarowi RTC z kalendarzem. Możliwości konfiguracyjne pozwalają na taktowanie zegara RTC z głównego zegara systemu (jeśli ma on odpowiednią częstotliwość) lub kwarcowego oscylatora zegarkowego o częstotliwości 32768 Hz,
– wiele zaimplementowanych konfigurowalnych sterowników m.in.: portu szeregowego, przetwornika ADC, portu SPI, pamięci EEPROM, alfanumerycznego wyświetlacza LCD.

Czym zajmuje się system operacyjny

Jak nadmieniono wcześniej, głównym zadaniem systemu operacyjnego jest zarządzanie pracą aplikacji uruchomionych w tym systemie. Składają się na to między innymi następujące elementy:

1. Podział czasu pracy kontrolera pomiędzy zadania.

Dzieląc czas pracy kontrolera na jednostkowe interwały (w najprostszym przypadku) i przydzielając je kolejno uruchomionym programom umożliwiamy symultaniczną ich pracę. Jeśli owo przełączanie zadań następuje odpowiednio szybko (najmniej kilkanaście razy na sekundę) otrzymujemy wrażenie równoległego działania wszystkich uruchomionych w systemie aplikacji. Ideę tę przedstawiono na **rys. 1**. W chwili startowej wykonywane jest zadanie 1. Po określonym czasie system przerywa jego pracę i przekazuje w posiadanie procesor kolejnemu oczekującemu zadaniu 2. Po chwili czas procesora jest przydzielony zadaniu 3, itd. Taka karuzela kręci się udostępniając moc obliczeniową procesora kolejno wszystkim potrzebującym zadaniom. Taki algorytm przydzielania czasu procesora zadaniom został nazwany **algorytmem karuzelowym**.



Rys. 2. Algorytm działania przykładowego programu

Wszystkie wykonywane przez mikrokontroler programy mają bardzo podobny charakter pracy przedstawiony na **rys. 2**. Jest to praca „impulsowa" rozdzielona okresami „bezczynności", w których procesor odmierza zadany czas lub oczekuje na jakieś zdarzenie (nadejście znaku z określonego portu, naciśnięcie przycisku itp.). W tradycyjnie działającym systemie mikroprocesorowym czasy oczekiwania to najczęściej wykonywanie w pętli pustych (instrukcji „nop") z ewentualnym sprawdzaniem stanu flag bądź rejestrów. Gdy w systemie mikroprocesorowym zostanie uruchomiony system operacyjny, czas bezczynności lub czas oczekiwania na zadanie X jest spożytkowany przez zadanie Y, które w tym momencie oczekuje właśnie na możliwość skorzystania z mocy obliczeniowej procesora. Sytuację taką ilustruje **rys. 3**.

Początkowo wykonywane jest zadanie 1 do momentu realizacji opóźnienia o czasie trwania 10 ms. Następnie moc obliczeniowa procesora zostaje udostępniona zadaniu 2 na czas 4 ms i kolejnemu zadaniu 3 na czas 6 ms. Po upływie odmierzanego dla zadania 1 czasu (10 ms) jest ono ponownie wykonywane.

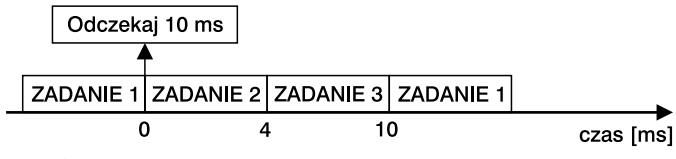
2. Odmierzanie czasu.

Jest to zadanie systemu operacyjnego ściśle związane z realizacją poprzedniej funkcji. Jeśli w systemie jest zaimplementowane odmierzanie czasu, a nie tylko jego interwałów, możliwa jest prosta realizacja zegarów czasu rzeczywistego (RTC).

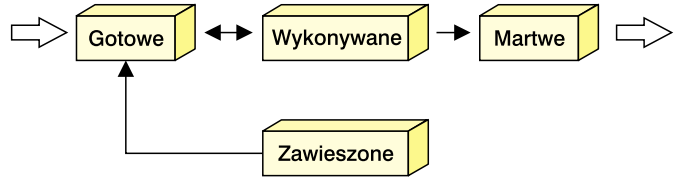
3. Zarządzanie zadaniami (kolejkowanie i szeregowanie).

Funkcja ta wynika bezpośrednio z założenia wielozadaniowości. Ponieważ w danej chwili zadania mają różne potrzeby (wykonywanie programu, odmierzanie czasu, oczekiwanie na wynik z przetwornika ADC, wysyłanie danych przez UART, itd.), system powinien w miarę możliwości optymalnie spełniać „zachcianki" poszczególnych zadań. Jest to realizowane poprzez tworzenie kolejek zadań uszeregowanych (lub nie) według określonego kryterium (np. priorytet – ważność zadania). W danym momencie tylko jedno zadanie jest wykonywane, pozostałe natomiast czekają w kolejkach na określone zdarzenie, po wystąpieniu którego zostaną przesunięte do innej kolejki, w szczególności do kolejki zadań oczekujących na zasób o nazwie jednostka centralna. Dla przykładu w komercyjnym systemie operacyjnym czasu rzeczywistego QNX każde zadanie może znajdować się w jednej z kolejek (w jednym ze stanów): *gotowe*, *zawieszone*, *wykonywane*, *martwe*. Możliwe przejścia pomiędzy kolejkami (stanami) przedstawiono **rys. 4**.

Każde wystartowane zadanie znajduje się w stanie *gotowe*. Zadanie zakończone przechodzi w stan *martwe*. Zadanie aktualnie posiadające procesor znajduje się w stanie *wykonywane*, natomiast zadanie nie mogące kontynuować swojego działania (oczekujące na zdarzenie) jest przesuwane do kolejki *zawieszone*.



Rys. 3. Podział czasu pracy kontrolera pomiędzy zadania oczekujące



Rys. 4. Wędrówka zadań w systemie QNX

4. Zarządzanie pamięcią operacyjną.

Ponieważ w systemie równolegle i asynchronicznie pracuje wiele aplikacji, a każda z nich może w dowolnej chwili korzystać z pamięci, konieczna jest synchronizacja podczas dostępu do obszarów współdzielonych, a także ochrona obszarów prywatnych. Na pewno niejeden z czytelników spotkał się z komunikatem od systemu Windows o treści: „Program wykonał nieprawidłową operację i zostanie zamknięty...". Najczęstszą tego przyczyną jest właśnie próba nieuprawnionego dostępu do pamięci.

5. Obsługa układów peryferyjnych (tworzenie interfejsu dla aplikacji).

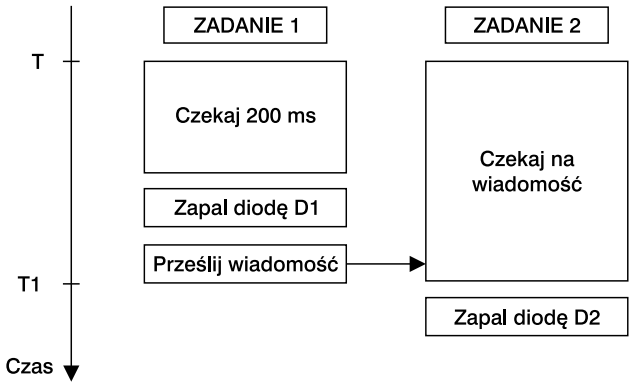
Wiąże się to z udostępnianiem poszczególnym zadaniom urządzeń we/wy w sposób wykluczający jednoczesne sterowanie układu peryferyjnego przez więcej niż jedno zadanie. Oczywistym jest, że taka sytuacja (brak synchronizacji) nie pozwala żadnemu z zadań poprawnie obsłużyć określonego zasobu. Dostęp do układów sprzętowych powinien odbywać się wyłącznie za pośrednictwem funkcji bibliotecznych systemu operacyjnego.

6. Udostępnienie interfejsu komunikacji między zadaniami.

Zwykle istnieje konieczność komunikowania się (przekazywania danych) zadań uruchomionych w systemie. Powinien zatem istnieć protokół umożliwiający niezawodną realizację tych potrzeb.

7. Synchronizacja zadań.

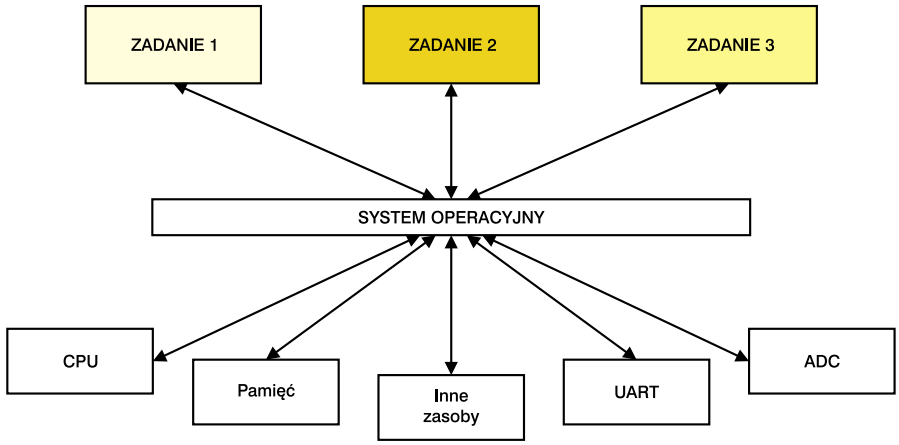
Ponieważ zadania w systemie z definicji są wykonywane niezależnie od siebie (asynchronicznie), może zajść konieczność zsynchronizowania ich pracy. Synchronizacja polega na uzależnieniu czasowym wykonania określonego działania od określonego zdarzenia. Na **rys. 5** przedstawiono omawiany proces. W przykładzie tym synchronizowane jest zadanie 2 do zadania 1. W chwili początkowej T wykonywane jest zadanie 1, a zadanie 2 oczekuje na komunikat synchronizujący. W chwili T1 następuje synchronizacja zadań, czego wizualnym dowodem jest jednoczesne zapalenie diod D1 i D2.



Rys. 5. Idea synchronizacji zadań

System operacyjny czasu rzeczywistego (*Real Time Operating System*)

RTOS to system, który w przewidywalnym czasie dostarczy oczekiwany rezultat działania (odpowiedź). W systemie czasu rzeczywistego nie tylko jest istotny otrzymany wynik, niemniej ważny jest czas jego otrzymania. Istnieją sytuacje, gdy otrzymany zbyt późno nawet poprawny rezultat jest bezużyteczny. Załóżmy (bez zbytniego wnikania w szczegóły zagadnienia), że pracą koparki zamiast operatora steruje mikrokontroler, a przesuwana łyżka zbliża się do szklanej ściany z prędkością 1 m/s. Gdy znajduje się 1 m od przeszkody rozpoczynamy obliczanie wartości opóźnienia jakie należy nadać obiektowi, aby wyhamować przed ścianą. Jeśli poprawną odpowiedź otrzymamy po czasie dłuższym lub równym 1 sekundzie, to możemy być pewni, że łyżka zatrzyma się nad kupą szkła z rozbitej ściany. Po tej uproszczonej analizie problemu chyba każdy Czytelnik rozumie czym jest czas rzeczywisty.



Rys. 6. Schemat systemu mikroprocesorowego wyposażonego w system operacyjny

Istnieje podział systemów czasu rzeczywistego na *soft (*miękkie*)* i *hard (*twarde*) real time systems*. Od tych pierwszych wymaga się dostarczenia odpowiedzi w średnim założonym czasie. Niewielkie opóźnienie jest tolerowane. Natomiast od systemów o twardych wymaganiach czasowych żąda się dostarczenia wyniku w nieprzekraczalnym terminie. Jakiekolwiek opóźnienie powoduje katastrofę.

Klocki poskładane

Na **rys. 6** został przedstawiony schematycznie system mikroprocesorowy działający pod kontrolą systemu operacyjnego. Można zauważyć, że poszczególne zadania (aplikacje użytkownika) nie mają bezpośredniego dostępu do zasobów sprzętowych, lecz wyłącznie za pośrednictwem systemu operacyjnego. Na żądanie każde z nich otrzymuje dostęp do określonego zasobu poprzez funkcje systemowe, pod warunkiem, że ów zasób nie jest aktualnie zajęty przez inne zadanie. W takim przypadku zadanie chcące skorzystać z zajętego zasobu musi poczekać na jego zwolnienie (w odpowiedniej kolejce). Wynika to bezpośrednio z funkcji realizowanych przez system przedstawionych powyżej.

Wady i zalety

Jakie korzyści płyną z zastosowania systemu operacyjnego? Oto kilka z nich:

- możliwość podzielenia funkcjonalności projektowanego firmware'u na niezależne bloki algorytmiczne realizowane w autonomicznych zadaniach,
- korzystanie z funkcji systemowych zwalnia nas z obowiązku bezpośredniej ingerencji w reje-

QNX od środka

W sklepie internetowym *sklep.avt.pl* jest dostępna książka „Systemy czasu rzeczywistego QNX6 Neutrino" autorstwa Jędrzeja Ułasiewicza, poświęcona jednemu z najpopularniejszych profesjonalnych systemów operacyjnych czasu rzeczywistego.
Jej kod handlowy: KS-280104.

stry mikrokontrolera, co ułatwia i przyspiesza pracę,
– pisząc kod zadania możemy zapomnieć o istnieniu innych zadań (chyba, że istnieje potrzeba ich współpracy) i liniowo implementować algorytm,
– przenośność oprogramowania pomiędzy różnymi platformami sprzętowymi; jedynym wymaganiem jest „przepisanie" (istnienie) systemu operacyjnego pracującego z danym procesorem,

Dla zachowania równowagi kilka wad:
– jeśli system mikroprocesorowy ma działać z zainstalowanym systemem operacyjnym, to konieczne jest posiadanie samego systemu operacyjnego – można go zakupić jako produkt komercyjny, skorzystać z darmowego lub napisać własny,
– jak każdy działający program, system operacyjny potrzebuje mocy obliczeniowej procesora do wykonywania powierzonych mu zadań, pomimo iż jego praca nie ma bezpośredniego związku z algorytmem realizowanym przez całą aplikację. Mamy dla przykładu do dyspozycji układ o wydajności 10 MIPS. Uruchamiamy systemem operacyjny, który średnio zużywa 10% jego mocy obliczeniowej. Dla właściwego oprogramowania pozostaje zatem maszyna o wydajności 9 MIPS.
– w przypadku mikrokontrolerów o niewielkim rozmiarze pamięci (ROM czy RAM) może okazać się, że po uruchomieniu systemu nie pozostanie wiele wolnego miejsca dla aplikacji.

W kolejnych częściach cyklu będziemy poszerzać wiedzę o systemie KaRTOS. Autor zachęca Czytelników do kształtowania ich treści. Prosimy o bezpośrednie kierowanie życzeń na jego adres e–mailowy. Czy zatem jesteśmy zainteresowani teorią działania, czy raczej wolelibyśmy, aby były prezentowane przykładowe aplikacje i projekty dla systemu KaRTOS. Inne uwagi również mile widziane.

Mariusz Żądło
iram@poczta.onet.pl