Systemy operacyjne systemów wbudowanych

Piotr Szymczyk

W artykule przedstawiono problem wyboru właściwej metody implementacji aplikacji dla systemu wbudowanego (z użyciem wielkiej pętli lub z systemem operacyjnym czasu rzeczywistego). Przedstawiono sposób porównania i wyboru systemu operacyjnego najbardziej odpowiedniego dla tworzonego systemu wbudowanego. Artykuł zawiera również charakterystykę dwóch reprezentatywnych systemów operacyjnych czasu rzeczywistego – komercyjnego VxWorks oraz bezpłatnego FreeRTOS.

Pojawienie się mikroprocesorów, a następnie mikrokontrolerów oraz układów SoC (System on Chip) umożliwiło wykorzystanie systemów wbudowanych w niemalże każdym współczesnym urządzeniu. Za przykład mogą służyć produkowane obecnie samochody, gdyż zawierają one ponad 70 specjalizowanych mikroprocesorów [1] odpowiedzialnych za pracę silnika, systemu kontroli trakcji, świateł, klimatyzacji itd. Przykładem może być też samolot F16, który jest niestabilny aerodynamicznie i nie mógłby wznieść się w powietrze bez systemów sterujących jego lotem. Szereg urządzeń powszechnego użytku jest wyposażonych w systemy wbudowane sterujące ich pracą, co umożliwia miniaturyzację i obniżenie ceny oraz w znaczący sposób zwiększa funkcjonalność i atrakcyjność dla użytkownika.

Systemy wbudowane wyróżniają się kilkoma cechami spośród innych systemów, a mianowicie są przede wszystkim przeznaczone do realizacji specyficznych zadań (w przeciwieństwie do komputerów PC, które mogą wykonywać różne prace), zazwyczaj spełniają wymagania czasu rzeczywistego [1], mogą być implementowane na szerokiej gamie procesorów i architektur procesorowych, a ich zastosowanie znacząco obniża koszty wytwarzania i utrzymania danego produktu. Wiele urządzeń trudno byłoby sobie w tej chwili wyobrazić bez systemów wbudowanych. Liczba takich urządzeń stale rośnie i w niedalekiej przyszłości niemal każde z nich będzie wyposażone w system wbudowany.

Oprogramowanie systemów wbudowanych

Oprócz zwykłych wymagań poprawności wykonywanych obliczeń, systemy wbudowane muszą spełniać stawiane im często wymagania czasowe. Mogą to być miękkie lub twarde warunki, zależnie od miejsca zastosowania i od funkcji jaką pełnią. Twarde wymagania czasowe przewidują wykonanie zadania obliczeniowego w pewnym ściśle określonym oknie czasowym. Zbyt późne wyznaczenie sterowania może prowadzić do błędnego działania systemu i zdyskwalifikować taką aplikację. Przykładem może być system sterowania i kontroli parametrów lotu w samolocie, gdzie wyznaczenie sterowania po upływie wymaganego czasu może doprowadzić do katastrofy. Mniej rygorystyczne wymagania – miękkie warunki czasowe – nakładają na system ograniczenia co do wartości średnich czasu trwania procesu obliczeniowego, przykładem takich systemów mogą być drukarki, w których oczekuje się, że średnia wydajność wynosi na przykład 3 strony wydruku na minutę, ale nie stanie się nic tragicznego, jeśli czasami drukarka będzie drukowała jedynie 2 strony na minutę.

Podstawowe sposoby realizacji zadań sterowania w czasie rzeczywistym to uaktywnianie zadania po określonym czasie (ang. *clock driver*) lub po wystąpieniu danego zdarzenia (ang. *event driver*).

Większość wbudowanych aplikacji, w tym aplikacje czasu rzeczywistego bazują na wykonywaniu zadań w tak zwanej dużej pętli, w której poszczególne zadania są uruchamiane sekwencyjnie jako kolejne procedury wewnątrz tej pętli. Takie rozwiązanie jest mało elastyczne i nie nadaje się do implementowania bardziej złożonych i wymagających systemów oraz nie daje możliwości obsługi asynchronicznych zdarzeń w czasie rzeczywistym. Programowanie z użyciem dużej pętli ma jednak niezaprzeczalną zaletę, mianowicie jest proste i przejrzyste, nadaje się do stosunkowo prostych aplikacji realizujących pojedyncze funkcje.

Ponieważ rozwój technologii doprowadził do pojawienia się sprzętu o dużych możliwościach za minimalną cenę, wzrasta zainteresowanie wykorzystaniem tego typu układów w bardziej zaawansowanych aplikacjach, w tym w systemach wbudowanych czasu rzeczywistego. Dla takich zastosowań programowanie z użyciem dużej pętli jest niemożliwe lub bardzo

dr inż. Piotr Szymczyk – Katedra Automatyki, Akademia Górniczo-Hutnicza

uciążliwe. Jedynym uniwersalnym rozwiązaniem jest zastosowanie wielozadaniowego systemu operacyjnego, zawierającego mechanizm wywłaszczania, który umożliwia stworzenie aplikacji składającej się z wielu zadań pracujących niezależnie, posiadających swoją ważność z punktu widzenie czasu rzeczywistego oraz komunikujących się ze sobą oraz z otoczeniem [2, 5, 9]. Należy jednak zawsze mieć na uwadze, że sprzęt tutaj stosowany, np. mikrokontrolery, ma i będzie miał, mimo znacznego rozwoju, szereg ograniczeń i w związku z tym nie można stosować systemu operacyjnego czasu rzeczywistego (SOCR) o zbyt wysokich narzutach i wymaganiach dotyczących zasobów (system operacyjny dla własnych potrzeb nie może zabierać zbyt dużo pamięci i obciążać procesor), ponieważ uniemożliwi to realizację wielu aplikacji [11].

Rys. 1. Porównanie metod programowania za pomocą dużej pętli i z użyciem SOCR

Systemy operacyjne systemów wbudowanych

Wybór odpowiedniego systemu operacyjnego, tak jak wybór platformy sprzętowej, jest jedną z najważniejszych decyzji, które bezpośrednio rzutują na parametry projektowanego systemu wbudowanego. Nie ma rozwiązań uniwersalnych i konkretny wybór powinien być podyktowany wymaganiami stawianymi wobec wyrobu końcowego (np. ilość miejsca na sprzęt, środowisko pracy, parametry czasowe, złożoność problemów obliczeniowych, ilość danych, liczba i rodzaj wejść i wyjść), możliwościami zamawiającego oraz zespołu wykonującego projekt (np. koszt jednostkowy produktu, całkowity koszt projektu, znajomość danych technik programowania, języków programowania czy systemów operacyjnych).

Dodatkowo należy uwzględnić fakt, że dla każdej platformy sprzętowej istnieje szczególnie dobrze zoptymalizowane jądro systemu operacyjnego czasu rzeczywistego – należy więc wybrać odpowiedni sprzęt lub odwrotnie, mając wybrany system operacyjny dokonać właściwego wyboru platformy sprzętowej. Kolejny dość często spotykany problem to ścisły związek systemu operacyjnego z kompilatorem języka (przeważnie języka C). Wybór systemu determinuje w takim przypadku wykorzystywane podczas tworzenia aplikacji konkretnego kompilatora, co może powodować konsekwencje na etapie pisania kodu.

W systemach wbudowanych można dokonać generalnego podziału systemów operacyjnych na systemy komercyjne (przykładem jest VxWorks firmy Wind River [8]) i systemy bezpłatne (takie jak FreeRTOS [6]). Zaletą systemów komercyjnych jest wielość narzędzi, wsparcie techniczne, bogata dokumentacja oraz szereg referencji. Oczywistą wadą są koszty środowiska do tworzenia aplikacji, wykształcenia specjalistów oraz koszty licencji na instalacje w produkcie docelowym. Z kolei systemy bezpłatne mogą być pod wieloma względami porównywalne z systemami komercyjnymi przy niebagatelnej zalecie w postaci braku opłat. Budzą one jednak czasami obawy co do jakości i bezpieczeństwa aplikacji. Obawy te są często nieuzasadnione i wynikają z braku wiedzy na temat tych systemów i z obiegowych opinii.

Aby wspomóc proces wyboru właściwego systemu operacyjnego czasu rzeczywistego, można zdefiniować następującą listę kwestii do rozstrzygnięcia [3], czy wybrany system operacyjny czasu rzeczywistego pasuje do projektowanego systemu wbudowanego.

Należy ustalić, czy SOCR:

- działa na rozważanej platformie sprzętowej
- umożliwia rozwijanie aplikacji w rozważanym języku programowania
- pracuje z posiadanym debuggerem sprzętowym, kompilatorem, assemblerem, linkerem i debuggerem kodu źródłowego
- ma zaimplementowane usługi, które będą wykorzystane w aplikacji, takie jak kolejki komunikatów, mechanizmy zegarowe, semafory, wspólna pamięć itp.
- ma odpowiednie algorytmy szeregowania procesów lub wątków
- zapewnia wystarczającą liczbę priorytetów procesów lub wątków
- ma mechanizm zabezpieczający przed powstawaniem zjawiska inwersji priorytetów
- zapewni odpowiednią skalowalność dla przewidywanej aplikacji, np. wielkość pamięci RAM, ROM
- zapewni odpowiednią szybkość działania (poprzez sprawdzenie odpowiednich wyników testów wydajnościowych często opracowywanych przez producentów sprzętu lub SOCR)
- obsługuje wymagane komponenty oprogramowania takie, jak: stos protokołów, serwisy komunikacyjne, bazy danych czasu rzeczywistego, serwisy webowe, biblioteki graficzne itd.
- ma drivery do rozważanych elementów sprzętowych
- dostarczy dodatkowe oprogramowanie narzędziowe, które wspomoże proces uruchamiania i testowania aplikacji
- zapewni zgodność z wymaganymi standardami
- oferuje odpowiednie wsparcie techniczne, szkolenia i konsultacje ze strony producenta
- dostarczy kod źródłowy czy kod obiektowy lub linkowalną bibliotekę
- ma jasną politykę licencyjną dotyczącą środowiska deweloperskiego oraz licencji SOCR na sprzęt docelowy, która jest zrozumiała i odpowiednia dla projektu

- ma odpowiednie pozytywne referencje
- gwarantuje właściwy czas wykonania projektu (ang. *time to market*)
- nie spowoduje przekroczenia całkowitego przewidywanego kosztu projektu?

Oczywiste jest, że dla każdego projektu są ważniejsze i mniej istotne parametry, dlatego też rozważane kwestie powinny mieć indywidualnie dobrane wagi umożliwiające całościową ocenę przydatności danego SOCR w projekcie.

VxWorks

VxWorks jest systemem operacyjnym czasu rzeczywistego firmy Wind River. Należy on do grupy nowoczesnych systemów operacyjnych opartych na koncepcjach wielozadaniowości, komunikacji między zadaniami, mikrojądra oraz procedur obsługi przerwań. Dysponuje także bogatym zestawem zaimplementowanych funkcji komunikacji sieciowej. VxWorks spełnia w dużej części wymagania określone w specyfikacji POSIX 1003.1b [10]. Jest dostępny dla bardzo wielu platform sprzętowych. Ma bardzo długą listę zastosowań. Pracuje w ponad 350 miliardach urządzeń. Był pierwszym komercyjnym systemem operacyjnym czasu rzeczywistego wykorzystanym przez NASA w misji na inną planetę (Mars Pathfinder).

Rys. 2. Środowisko skrośne rozwoju aplikacji dla systemu VxWorks

System VxWorks charakteryzuje się:

- efektywnym zarządzaniem zadaniami:
  - wielozadaniowość, nieograniczona liczba zadań
  - algorytm szeregowania z wywłaszczaniem oparty na priorytetach oraz algorytmie karuzelowym
  - szybkie deterministyczne przełączanie kontekstu
  - 256 poziomów priorytetów
- szybką i elastyczną komunikacją między zadaniami:
  - semafory binarne, całkowite, wzajemnego wykluczania oraz mechanizm dziedziczenie priorytetów
  - kolejki komunikatów
  - elementy standardu POSIX
  - współdzielona pamięć
- szybką i efektywną procedurą obsługi przerwań
- dynamicznym zarządzaniem pamięcią
- zegarem systemowym i dużymi możliwościami związanymi z zarządzaniem czasem
- sieciowością:
  - protokół TCP/IP
  - różne media: Ethernet, łącze szeregowe, szyna systemowa
  - gniazda
  - zdalny dostęp (rlogin, telnet), zdalne wywoływanie procedur (RPC), zdalne wykonywanie komend (rsh)
  - sieciowy system plików NFS, usługi ftp, tftp klient i serwer
  - protokoły: BOOTP, SNMP
- szybką i elastyczną obsługą wejścia-wyjścia oraz lokalnego systemu plików:
  - całkowicie przenośny system obsługi wejścia-wyjścia
  - standard POSIX – asynchroniczny system obsługi wejścia-wyjścia
  - obsługa urządzeń SCSI
  - rozszerzony system plików MS-DOS i RT-11
- możliwością rozwoju systemu docelowego:
  - zgodność ze standardem ANSI C i C++ oraz POSIX
  - interaktywna powłoka środowiska docelowego
  - dynamiczne łącznie funkcji
  - biblioteka ponad 1100 funkcji, w tym funkcje czasu rzeczywistego
  - możliwość ładowania systemu z pamięci ROM, lokalnego dysku lub poprzez sieć
- środowiskiem skrośnym rozwoju aplikacji
- bogatym zestawem dodatkowych pakietów oprogramowania:
  - Wind Foundation Classes – dodatkowy zestaw klas czasu rzeczywistego
  - VxVMI – rozszerzona obsługa pamięci
  - VxMP – rozszerzenie o wieloprocesorowość
  - BSP Porting Kit – pakiet umożliwiający przeniesienie systemu na dowolny sprzęt
  - WindPower Tools – zaawansowane pakiety WindView, Stethoscope, WindNavigator do testowania i uruchamiania aplikacji
  - środowiska graficzne: VX-Windows, Motif/X-Windows klient i serwer
  - RTGL Realtime Graphics Library
  - wirtualna maszyna Javy
  - VxSim – symulator środowiska docelowego.

Oprogramowanie dla systemu VxWorks przygotowywane jest na stacji roboczej (np. na komputerze PC pracującym pod kontrolą systemu operacyjnego MS Windows), gdzie jest do dyspozycji zintegrowane środowisko do tworzenia aplikacji – edytor kodu źródłowego, kompilator, debugger, symulator itd. Po skompilowaniu kod jest przenoszony na sprzęt docelowy i tam może być uruchomiony.

O szybkości tego systemu operacyjnego świadczą przykładowe pomiary czasów dokonane dla platformy docelowej wyposażonej w jeden procesor MPC8260 z zegarem o częstotliwości 50 MHz [4]:

- czas przełączania kontekstu 11 µs
- czas reakcji na przerwanie 98 µs
- czas pobrania semafora binarnego 13 µs
- czas propagacji komunikatu w kolejce komunikatów 118 µs.

Powyższe cechy oraz szereg uznanych referencji powodują, że jest to jeden z najpoważniejszych kandydatów, który powinien być brany pod uwagę przy wyborze komercyjnego SOCR.

FreeRTOS

FreeRTOS jest systemem operacyjnym czasu rzeczywistego dostępnym bez opłat. Wsparcie techniczne jest możliwe dzięki grupie użytkowników wzajemnie się wspierających poprzez forum dyskusyjne. Jest dostępny dla szeregu platform sprzętowych, a ich lista stale się poszerza. Został zaprojektowany specjalnie z myślą o wykorzystaniu w systemach wbudowanych i jest cały czas dynamicznie rozwijany. Ma skalowalne, przenośne jądro czasu rzeczywistego. FreeRTOS wymaga, aby zadania zostały zlinkowane z systemem operacyjnym i razem z nim wgrane na sprzęt docelowy (przekopiowane do pamięci urządzenia, na którym mają pracować). Typowe zadanie w tym systemie wygląda w sposób następujący [6]:

```
void vATaskFunction( void *pvParameters ) {
// kod inicjalizujący pracę zadania
for(;;) {
   // właściwy kod zadania
   // w niekończącej się pętli
        }
}
```

Najistotniejsze cechy systemu FreeRTOS to:

- wielozadaniowość z wywłaszczaniem, kooperacyjna wielozadaniowość, możliwość hybrydowej konfiguracji (wywłaszczania i kooperacji zadań)
- ma zarówno zadania jak i synchroniczne procesy współbieżne
- nieograniczona liczba zadań, które mogą być stworzone w systemie
- nieograniczona liczba priorytetów
- wiele zadań może mieć ten sam priorytet – brak restrykcji pod tym względem
- mechanizm wykrywania przepełnienia stosu
- bogaty zestaw metod komunikacji:
  - kolejki komunikatów
  - semafory binarne, całkowite, rekursywne
  - semafory wzajemnego wykluczania z dziedziczeniem priorytetów
- bardzo łatwy do przenoszenia na kolejne platformy sprzętowe ze względu na to, że system został napisany w języku C
- zaprojektowany z wyraźnym naciskiem na zminimalizowanie zapotrzebowania na wymagane zasoby (4,3 KB na ARM7), prostotę i łatwość użycia
- dostępny bezpłatnie kod źródłowy szeregu aplikacji wbudowanych
- środowisko skrośne rozwoju aplikacji
- możliwe użycie kompilatora GCC
- minimalny narzut na zasoby (ROM, RAM, moc procesora)
- dostępny jest szereg darmowych narzędzi wspomagających tworzenie i uruchomienie aplikacji
- dobra dokumentacja i szereg przykładowych aplikacji.

Obok FreeRTOS są rozwijane siostrzane projekty systemy OpenRTOS i SafeRTOS przez firmę WITTENSTEIN. OpenRTOS jest sprzedawaną komercyjnie wersją FreeRTOS zapewniającą wsparcie techniczne, konsultacje i pełną dokumentację. SafeRTOS ma certyfikat SIL3 RTOS zgodny z IEC 61508 (odpowiednik PN-EN 61508).

VxWorks a FreeRTOS

Opisane systemy w zasadzie są stosowane w rozłącznych obszarach zastosowań. Istnieje jednak pewien zbiór aplikacji wbudowanych, które mogą być alternatywnie implementowane na obu tych systemach.

W tabeli przedstawiono porównanie ważniejszych cech systemu VxWorks i FreeRTOS.

Podsumowanie

Tworząc aplikację dla systemów wbudowanych należy dokonać wyboru metody jej realizacji: albo za pomocą dużej pętli, albo z użyciem systemu operacyjnego czasu rzeczywistego. Każda z tych metod ma swoje niezaprzeczalne zalety i znane wady. Decyzja zależy od konkretnych potrzeb danej implementacji. Jeśli wymagane jest by aplikacja pracowała w systemie operacyjnym, to kolejnym krokiem jest wybór systemu czasu rzeczywistego, który będzie najbardziej odpowiedni. Wybór ten nie jest prosty i wymaga szeregu porównań, które będzie łatwiej przeprowadzić stosując zalecane powyżej analizy. Jeśli wybór pada na komercyjny system operacyjny, to warto rozważyć system VxWorks, w przypadku wybrania oprogramowania bezpłatnego, wartym polecenia jest system FreeRTOS.

Tab. Porównanie niektórych cech VxWorks i FreeRTOS

Cecha	VxWorks	FreeRTOS
Platformy sprzętowe	Bardzo duża liczba	Średnia liczba
Wielozadaniowość z wywłaszczaniem	TAK	TAK
Kooperatywna wielozadaniowość	NIE	TAK
Liczba poziomów priorytetów	256	Definiowana
Bogaty zestaw metod komunikacji zadań	TAK	TAK
Zabezpieczenie przed inwersją priorytetów	TAK	TAK
Środowisko skrośne rozwoju aplikacji	TAK	TAK
Symulator	TAK	NIE
Dostępne dodatkowe narzędzia wspomagające	TAK (płatne)	TAK (bezpłatne)
Wsparcie, konsultacje	TAK (płatne)	Forum internetowe
Referencje	Bardzo duża liczba poważnych referencji	Średnia liczba
Cena	Drogi (cena zależy od zastosowania)	Bezpłatnie

Bibliografia

1. Berger Arnold S.: Embedded Systems Design: An Introduction to Processes, Tools and Techniques, CMP Books, 2002.
2. Curtis Keith E.: Embedded Multitasking With Small Microcontrollers, Newnes (Elsevier Group), 2006.
3. Hawley G.: Selecting a Real-Time Operating System, Embedded Systems Programming, March, 1999.
4. Ip B.: Performance Analysis of vxworks and rtlinux, Raport COMS W4995-2, Columbia University, NY 2001.
5. Jerraya A. A. (Editor), Yoo S. (Editor), Wehn N. (Editor), Verkest D. (Editor): Embedded Soft--ware for SoC, Kluwer Academic Publishers, 2003.
6. http://www.freertos.org
7. http://www.highintegritysystems.com
8. http://www.windriver.com
9. Lamie Edward L.: Real-Time Embedded Multithreading: Using threadx and ARM, CMP Books, 2005.
10. Szymczyk P.: Systemy operacyjne czasu rzeczywistego, Kraków, Wydawnictwa AGH, 2003.
11. Wilmshurst T.: Designing Embedded Systems with PIC Microcontrollers, Newnes (Elsevier Group), 2007. ■