AUTOMATYKA • 2008 • Tom 12 • Zeszyt 2

Dariusz Marchewka*, Marcin Piątek*

Stosowanie systemów wbudowanych do sterowania robotami mobilnymi

1. Wstęp

1.1. Ogólna budowa systemów wbudowanych

W związku z rozwojem elektroniki i systemów komputerowych sterowanie urządzeniami mechaniczno-elektrycznymi stało się relatywnie proste. Komputery i mikrokomputery spotyka się w każdej dziedzinie życia: począwszy od przenośnych urządzeń do odtwarzania muzyki i filmów, poprzez telefony komórkowe, systemy nawigacji satelitarnej, sterowniki zarządzające pracą samochodowych układów, aż po wyspecjalizowane systemy sterujące zaawansowanymi procesami technologicznymi.

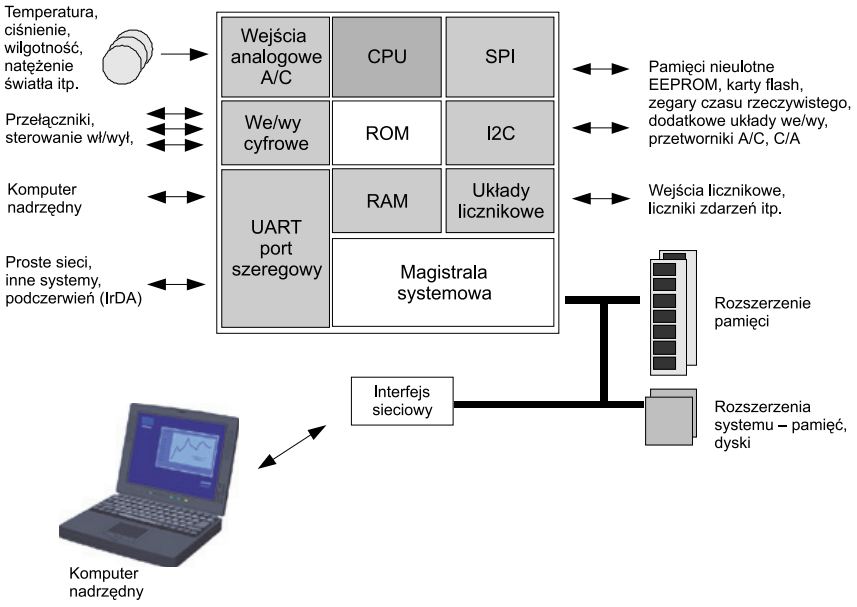
Każde z wymienionych urządzeń, pomimo iż wykonują różne zadania, łączy jedna wspólna cecha: zostały zaprojektowane i zaprogramowane do wykonywania jednego konkretnego zadania. Wykorzystanie do innych celów oczywiście jest możliwe, lecz najczęściej wiąże się z kosztowną przeróbką. Właśnie takie sterowniki w literaturze [7] nazywa się systemami wbudowanymi (*embedded*) zwykle dlatego, że są urządzeniami kompaktowymi o niewielkich wymiarach i niskim zużyciu energii. Inną cechą szczególną takich systemów jest niezawodność działania. Wykonane są jako dedykowane układy elektroniczne, przy których projektowaniu położono szczególną uwagę na długotrwałą i nieprzerwaną pracę niewymagającą praktycznie żadnej obsługi operatora (poza włączeniem i ewentualnym wyłączeniem).

Najprostsze systemy, stosowane w sprzętach powszechnego użytku, zawierają pojedynczy mikrokontroler z dedykowanym oprogramowaniem. Najbardziej skomplikowane zawierają wiele procesorów pracujących najczęściej pod kontrolą dedykowanych systemów operacyjnych i stosuje się je w robotyce, telekomunikacji, przemyśle samochodowym i lotniczym.

Systemy wbudowane mają ograniczoną do minimum komunikację z użytkownikiem. Często jest to jeden lub kilka przycisków do wyboru trybu pracy oraz kilka diod LED (*Light-*

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

-*Emitting Diode*) lub prosty wyświetlacz LCD (*Liquid Crystal Display*) do sygnalizacji aktualnego stanu systemu. Zarówno proste systemy, jak i te złożone mogą być łączone ze sobą lub komputerami stacjonarnymi w celu wymiany danych lub przeprogramowania. Ogólną strukturę systemu wbudowanego przedstawia rysunek 1.



Rys. 1. Ogólna struktura systemu wbudowanego

Głównym elementem prostego systemu wbudowanego jest mikrokontroler wyposażony w przetworniki A/C, C/A, we/wy cyfrowe, magistrale SPI, I2C, USB, port szeregowy, magistralę adresową/danych, pamięć wewnętrzną danych RAM i pamięć programu ROM (FLASH). Do wejść cyfrowych i analogowych można podłączyć czujniki stykowe, czujniki wielkości nieelektrycznych (np. temperatury, wilgotności, natężenia światła itp.). System można rozbudować przez dołączenie do istniejących magistral dodatkowych kart pamięci, wyświetlaczy. Należy jednak zwrócić uwagę na to, iż aby system był niezawodny liczba urządzeń peryferyjnych powinna być ograniczona [3].

Szczególnymi urządzeniami, w których z powodzeniem można stosować systemy wbudowane, są roboty mobilne. Przez pojęcie „robota mobilnego" w kontekście niniejszego artykułu należy rozumieć robota autonomicznego, jeżdżącego lub kroczącego, który przeznaczony jest do wykonywania określonych zadań, np. eksploracji nieznanego środowiska. Przykładem takiego robota może być *Sojourner*, bezzałogowy pojazd misji *Mars Pathfinder* [9] wykorzystywany do badania skał marsjańskich. Bardziej „przyziemnym" przykładem autonomicznego robota mobilnego może być automatyczny odkurzacz *iRobot Roomba* [10].

2. Część sprzętowa

Jak już wspomniano we wstępie, systemy wbudowane ze względu na swoje rozmiary, niski pobór energii i niezawodność doskonale nadają się do sterowania robotami mobilnymi. Stopień złożoności systemu zdeterminowany jest konstrukcją i funkcjonalnością robota. W przypadku robota kroczącego możliwe są dwa rozwiązania. Pierwsze to zastosowanie jednego mikrokontrolera i dołączenie do niego wszystkich elementów wykonawczych i czujników. Drugie to zastosowanie centralnego układu sterowania i lokalnych sterowników bezpośrednio połączonych z układami wykonawczymi i czujnikami. Zarówno jedno, jak i drugie rozwiązanie, ma wady i zalety. W przypadku mobilnego robota kroczącego, na jedną kończynę o trzech stopniach swobody przypadają co najmniej trzy układy wykonawcze i trzy czujniki położenia. Jeżeli przemnożymy tę liczbę przez liczbę nóg i uwzględnimy liczbę sygnałów sterujących do każdego układu wykonawczego i czujnika, to w pierwszym rozwiązaniu, w najgorszym przypadku, może nam zabraknąć wolnych pinów mikrokontrolera, aby podłączyć wszystkie urządzenia. Wadą takiego rozwiązania jest duża liczba przewodów łączących elementy wykonawcze i czujniki ze sterownikiem, a tym samym większe prawdopodobieństwo uszkodzenia mechanicznego.

W drugim rozwiązaniu lokalne sterowniki, kontrolujące pracę np. pojedynczej kończyny, połączone są z jednostką centralną za pomocą wybranej magistrali danych (np. I2C, SPI). W tym przypadku jedynym ograniczeniem jest maksymalna liczba urządzeń typu *slave* dla zastosowanego połączenia. W tego typu rozwiązaniach stosuje się najczęściej dwu- lub trzyprzewodowe magistrale, co w znacznym stopniu zmniejsza liczbę przewodów w porównaniu do pierwszego rozwiązania. Rozwiązania rozproszone wymagają większego nakładu pracy przy tworzeniu oprogramowania sterującego niż w rozwiązaniach scentralizowanych. Do prawidłowego działania takiego systemu konieczne jest oprogramowanie wszystkich lokalnych węzłów i jednostki nadrzędnej – w przypadku konstrukcji sześcionożnej przy założeniu, że każdą z nóg steruje jeden węzeł musi powstać siedem niezależnych programów, wymieniających odpowiednie dane między sobą. W przypadku systemu jednoprocesorowego powstanie jeden program, jednak o większej złożoności, niż program jednostki centralnej systemu rozproszonego.

2.1. Komponenty systemu wbudowanego

Mikrokontroler

„Sercem" systemu zarówno jednoprocesorowego, jak i rozproszonego jest mikrokontroler. Mikrokontrolerem nazywamy procesor wyposażony w podstawowe układy wejścia/wyjścia, podręczną pamięć danych RAM, pamięć programu ROM/EEPROM/FLASH, magistralę adresową/danych, porty komunikacji szeregowej UART, I2C, SPI itp. Praktycznie każdy producent procesorów posiada w swojej ofercie kilka typów mikrokontrolerów – począwszy od prostych 8-bitowych mikrokontrolerów, poprzez 16/32-bitowe architektury z rdzeniem ARM7 lub ARM9, aż po specjalizowane procesory DSP.

Ze względu na różnorodność dostępnych układów wybór odpowiedniej architektury jest trudny.

Wyboru docelowego mikrokontrolera dokonuje się najczęściej poprzez analizę kilku założeń:

- złożoności planowanego algorytmu (czas wykonywania algorytmu),
- ilości i rodzaju planowanych czujników i układów wykonawczych,
- koszty układów.

W przypadku systemów sterowania wykonujących skomplikowane obliczenia mikrokontroler powinien być wyposażony w jednostkę do wykonywania obliczeń zmiennoprzecinkowych (FPU – *Floating Point Unit*). W takich przypadkach najczęściej nie wystarcza wewnętrznej pamięci RAM do realizacji obliczeń bądź przechowywania danych. Konieczne staje się rozszerzenie tej pamięci. Problem pojawia się w sytuacji, gdy system potrzebuje więcej pamięci niż można jej podłączyć do magistrali adresowej mikrokontrolera. Jeżeli mikrokontroler nie jest wyposażony w układ zarządzania pamięcią (MMU – *Memory Management Unit*), to obsługa dodatkowej pamięci spada na programistę. W mikrokontrolerach wyposażonych w MMU układ ten dokonuje translacji odwołań do logicznych adresów na adresy w fizycznej pamięci. W takim przypadku nie ma znaczenia, jakiego typu jest dodatkowa pamięć.

Pamięć i układy we/wy

W systemach wbudowanych jako nośnik danych wykorzystuje się w pierwszej kolejności wewnętrzną pamięć RAM lub EEPROM mikrokontrolera. Dodatkowo do systemu można podłączyć poprzez odpowiednią magistralę karty pamięci FLASH (np. SD, MMC, xD). Jest to wygodne rozwiązanie, gdyż dzięki temu można zgromadzone dane łatwo przenosić.

Jak wspomniano wcześniej, mikrokontrolery mają wbudowane układy wejścia/wyjścia. Należą do nich najczęściej wejścia i wyjścia cyfrowe, oraz, w zależności od typu mikrokontrolera, wejścia i wyjścia analogowe z wbudowanych w układ przetworników A/C i C/A. Zamiast wyjść analogowych często spotyka się wyjścia cyfrowe typu PWM (*Pulse Width Modulation*) – modulacja szerokością impulsu. W takich przypadkach sygnał PWM podawany jest na element wykonawczy bezpośrednio lub przez filtr dolnoprzepustowy.

Do komunikacji z zewnętrznym układem bądź systemem może być wykorzystywany port szeregowy, port USB lub coraz częściej Ethernet. Praktycznie większość z dostępnych na rynku mikrokontrolerów zawiera co najmniej dwa lub wszystkie wymienione powyżej mechanizmy komunikacji.

3. Część programowa

Zastosowanie mikrokontrolera lub innego układu programowalnego (np. FPGA) w omawianym układzie umożliwia stosunkowo łatwą i szybką zmianę algorytmu bądź jego parametrów. Pozwala to na szybkie prototypowanie części programowej sterowników i łatwe strojenie nastaw, niewymagające wprowadzania drogich i pracochłonnych zmian sprzętowych. Dużą zaletą jest także możliwość zastosowania tej samej części sprzętowej w całej rodzinie urządzeń, uzupełniając jedynie oprogramowanie.

3.1. Dedykowane systemy sterowania

Rolą oprogramowania w systemie wbudowanym jest realizacja pewnego algorytmu, co w większości przypadków sprowadza się do realizcji prostego cyklu operacji. Całość rozpoczyna się od odczytu danych wejściowych. Następnie dochodzi do obróbki tych danych i realizacji dyskretnej funkcji regulatora, aby wyliczyć wartości wyjściowe. Algorytm kończy się przesłaniem wyniku do wyjść układu. Cykl taki może być zarówno wyzwalany po upływie pewnego interwału czasowego (wyzwalany zegarem), jak i przez zmianę wejścia sterownika spowodowaną zajściem charakterystycznych warunków w układzie (wyzwalany przerwaniem zewnętrznym).

Im bardziej skomplikowany układ sterowania, tym więcej różnego typu cykli będzie realizowane równocześnie. Na przykład sterownik robota o trzech stopniach swobody będzie realizował równocześnie trzy algorytmy dla każdego z silników sterujących, a dodatkowo przynajmniej jeden związny z odpowiednią reakcją na włączenie ograniczników ruchu (krańcówek) i jeden związany z archiwizacją/wizualizajcą danych o ruchu robota. Praktyczna realizacja tego typu systemu będzie potrzebowała kilka dodatkowych zadań związanych z obliczeniem wyższych poziomów algorytmu sterowania.

Nasuwa się więc prosty wniosek: większa złożoność zadania sterującego wymaga lepszego wsparcia wielozadaniowości w rozwijanym oprogramowaniu.

W stosunkowo prostych przypadkach świetnie spisują się układy bazujące na mikrokontrolerach bez systemu operacyjnego [2]. Oprogramowanie tego typu jest realizowane w języku assembler takiego mikrokontrolera lub w coraz popularniejszym języku C (nawet na mikroprocesorach ośmiobitowych). Czym mniejsza skala problemu, tzn. czym mniejsza liczba zadań, które musimy realizować równolegle i mniejsza ich złożoność obliczeniowa, tym mniejszy i tańszy mikrokontroler możemy zastosować. Należy tutaj wspomnieć, że takie rozwiązania są często najszybsze i najbardziej precyzyjne, ze względu na fakt, iż na oprogramowanie składa się stosunkowo mało kodu źródłowego implementowanego w krótkim czasie przez małą liczbę programistów.

3.2. Systemy operacyjne w układach sterowania

Po przekroczeniu pewnego progu złożoności oprogramowania (patrz. 3.1) implementacja całości algorytmu sterującego bez dobrego wsparcia wielozadaniowości, odpowiedniego zarządzania pamięcią itp. staje się nieopłacalna i niezwykle pracochłonna. W rozbudowanych sterownikach stosuje się systemy operacyjne, które gwarantują wsparcie dla tych elementów. Pozwalają programistom na wykorzystanie w swojej pracy pewnych abstrakcji realizowanych przez system operacyjny. Typowym przykładem takiej abstrakcji jest wielozadaniowość tzn. możliwość implementacji i pseudorównoległego uruchamiania wielu zadań. W rzeczywistości dla każdego istniejącego w systemie mikroprocesora tworzona jest kolejka zadań, zgodnie z którą są one uruchamiane. Abstrakcja zadania ułatwia programiście pracę i przy odpowiednio niewielkich wymaganiach czasowych obiektu w stosunku do prędkości mikroprocesora pozwala interpretować zadania jako działające równolegle.

Systemy operacyjne zdecydowanie ułatwiają także proces zarządzania pamięcią w systemach sterowania. W przeciwieństwie do układów mikroprocesorowych bez systemu operacyjnego proces przydzielania i zwalniania pamięci odbywa się automatycznie i pozwala uniknąć wielu błędów związanych np. z dostępem do pamięci spoza zakresu przydzielonego dla danego zadania.

Poza tym system operacyjny najczęściej oferuje wsparcie dla ogromnej ilości sprzętu dostępnego na rynku, typu: moduły transmisji sieciowej, urządzenia do przechowywania dużych ilości danych (dyski twarde, karty pamięci flash itp.), urządzenia do komunikacji z użytkownikiem, a także standardowe protokoły wymiany danych.

Systemy operacyjne dla układów wbudowanych często są stosowane w przemyśle i są znane przeciętnemu użytkownikowi chociażby z telefonów komórkowych. Należą do nich:

- Symbian,
- Windows CE,
- różne rodzaje GNU/Linux.

W większości układów sterowania wymagania stawiane urządzeniom jako całości, a więc także oprogramowaniu, są dużo wyższe. To co charakteryzuje systemy stosowane w sterownikach to wymóg wypełniania ostrych kryteriów twardego czasu czasu rzeczywistego. Najpopularniejsze systemy tego typu stosowane w przemyśle to:

- QNX,
- WxWorks,
- różne rodzaje GNU/Linux zmodyfikowane, by spełniać kryteria RT.

Wszystkie te systemy charakteryzują się dużą skalowalnością, co pozwala je wykorzystywać jednocześnie na dużych stacjach roboczych, komputerach klasy PC i małych urządzeniach sterujących o ograniczonych zasobach. Efektywnie użytkownik może spotkać się z takim systemem w popularnych przełącznikach sieciowych, routerach czy drukarkach biurowych.

Możliwości systemu czasu rzeczywistego w zestawieniu z wbudowanym sterownikiem sprawiają, że jest on niemal idealnym kandydatem do zastosowania w złożonych aplikacjach z dziedziny robotyki.

System sterujący robotem rzadko wymaga czasów sterowania poniżej 1 ms, pseudozrównoleglenie procesów sterujących ułatwia pracę, a gotowe moduły obsługi protokołów sieciowych i innych metod komunikacji ułatwiają realizację wizualizacji i przechowywania danych.

3.3. Zastosowanie zmodyfikowanego systemu GNU/Linux w systemach sterowania

System operacyjny GNU/Linux posiada wszystkie wymieniane wcześniej zalety pozwalające na wykorzystanie go we wbudowanych układach sterujących, w tym układach dedykowanych dla robotyki [8]. Jego niezwykła skalowalność pozwala na instalację na większości z popularnych architektur mikroprocesorowy [11]. Dodatkowo posiada on szczególną zaletę: pełną dostępność kodu źródłowego i możliwość jego dowolnej modyfikacji zgodnej z popularną w środowisku Open Source licencją GPL. Cecha ta nabiera szczególnego znaczenia w przypadku projektów naukowych i dydaktycznych.

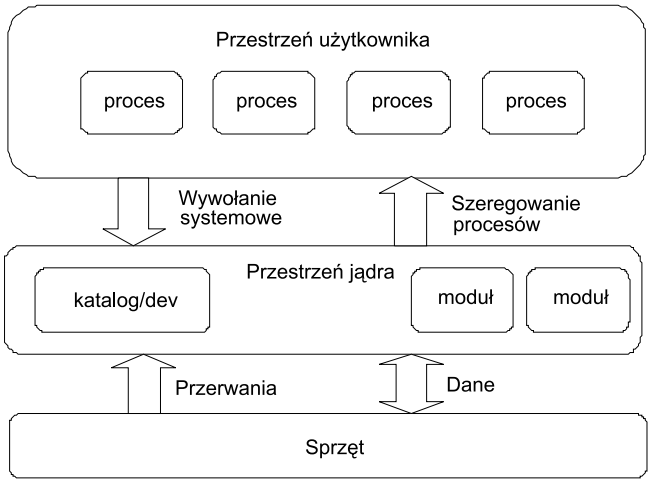
Architektura systemu GNU/Linux

System operacyjny GNU/Linux posiada architekturę typową dla systemów operacyjnych. Istnieje podział na tzw. przestrzeń jądra i przestrzeń użytkownika. W przestrzeni jądra realizowane są procedury jądra systemu operacyjnego i posiada ona bezpośredni dostęp do pamięci oraz magistral, a co za tym idzie – podłączonego sprzętu. Procedury dostępu do sprzętu w celu unifikacji zgromadzone są w komponencie jądra, które tradycyjnie nazywa się HAL (*Hardware Abstraction Layer*). Dzięki temu podziałowi proces implementacji wsparcia systemu GNU/Linux na nową platformę sprzętową ogranicza się do przepisania tego elementu.

W przestrzeni użytkownika realizowane są procesy, a komunikacja ze sprzętem i pamięcią odbywa się pośrednio poprzez kod jądra. Pamięć, którą widzi proces nie jest pamięcią fizyczną układu, ale pewnym obszarem w pamięci wirtualnej. Pozwala to na spełnianie kontroli nad dostępem do pamięci i zabezpiecza przed przypadkowym zapisem w obszarze przeznaczonym dla innego procesu. Operacja translacji adresu w pamięci wirtualnej do pamięci fizycznej musi być wspierana przez układ MMU (*Memory Management Unit*) fizycznie implementowany w większych mikroprocesorach. W przypadku jego braku można zastosować specjalną wersję systemu operacyjnego μCLinux, która została pozbawiona mechanizmu wirtualizacji pamięci.

Część małych mikroprocesorów nie posiada jednostki FPU (*Floating Point Unit*) fizycznie realizującej operacje arytmetyczno-logiczne na liczbach zmiennoprzecinkowych. System GNU/Linux potrafi emulować jej obecność (kosztem wydajności). Ma to duże znaczenie podczas realizacji pewnych, złożonych obliczeniowo algorytmów.

Dodatkowo system operacyjny GNU/Linux pozwala na implementację tzw. modułów jądra czyli komponentów, które możemy dołączyć do jądra i uruchomić w jego przestrzeni bez konieczności przeładowywania całego systemu operacyjnego [4]. Mechanizm ten ułatwia pracę i pozwala na szybszy rozwój np. sterowników obsługi sprzętu. Architektura systemu została przedstawiona na rysunku 2.



Rys. 2. Architektura systemu GNU/Linux

Czas rzeczywisty w systemie GNU/Linux

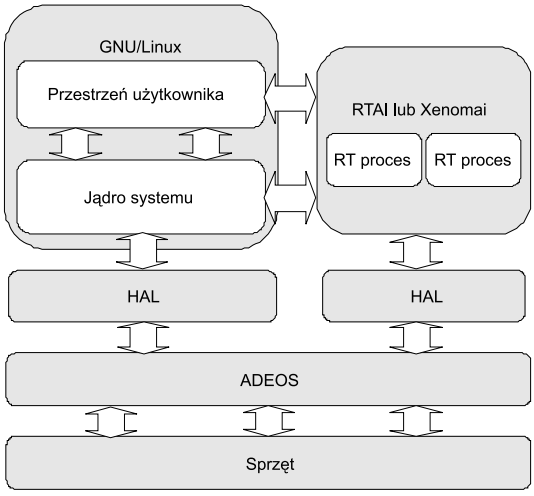
Od dłuższego czasu społeczność odpowiedzialna za implementację systemu GNU/Linux dostrzega potrzebę wprowadzenia zmian pozwalających na uruchamianie procesów z rygorem czasu rzeczywistego [1]. Klasyczna wersja systemu posiada już zaimplementowane odpowiednie mechanizmy, które pozawalają na uruchomienie procesów czasu rzeczywistego o miękkich wymaganiach czasowych. Istnieje także kilka implementacji, które pozwalają na uruchomienie zadań o twardych wymaganiach czasowych, np.:

- Montavista Linux,
- FMSLabs RT-Linux,
- RTAI (*Real Time Application Interface*),
- Xenomai.

Na nasze szczególne zainteresowanie zasługują dwa ostatnie jako projekty o otwartym kodzie źródłowym i udostępniane użytkownikom bez konieczności wnoszenia jakichkolwiek opłat.

Projekty RTAI i Xenomai

Projekty RTAI [12] oraz Xenomai [13] wykorzystują tę samą technikę realizacji wymogów czasu rzeczywistego, tak zwaną architekturę mikrojądra. Zamiast jednej warstwy pozwalającej na dostęp do sprzętu HAL istnieje ich kilka kontrolowanych przez mikrojądro projektu ADEOS. Jedna z nich to to warstwa HAL działającego w systemie GNU/Linuksa. Procesy czasu rzeczywistego są uruchamiane jednak poza Linuksem i posiadają od niego wyższy priorytet. Ich implementacja i uruchamianie odbywa się z poziomu Linuksa, a procesy czasu rzeczywistego mogą komunikować się zarówno ze zwykłymi procesami, jak i pomiędzy sobą. Odbywa się to z wykorzystaniem specjalnych mechanizmów IPC (*Inter-Process Communication*) [5].



Rys. 3. Architektura zmodyfikowanego systemu GNU/Linux, pozwalająca na uruchmianie zadań twardego czasu rzeczywistego

Zarówno w RTAI, jak i Xenomai istnieje możliwość uruchamiania procesów czasu rzeczywistego z poziomu przestrzeni jądra Linuksa, a także przestrzeni użytkownika Linuksa. Uruchamianie w przestrzeni jądra wymaga specjalnej implementacji (w postaci modułu) i narzuca kilka ograniczeń, ale daje lepsze wyniki czasowe. Rysunek 3 przedstawia schemat omawianej architektury.

Twórcy systemu RTAI kładą szczególny nacisk na wydajność czasową systemu, natomiast system Xenomai posiada niezwykle wygodny interfejs programistyczny oraz charakterystyczny mechanizm skór. Pozwala on na łatwą implementację nowego interfejsu programistycznego np. takiego który będzie kompatybilny z innym systemem czasu rzeczywistego. Wyniki pomiarów wydajności czasowej w systemie wykorzystującym komputer klasy PC można znaleźć w [6].

4. Planowane zastosowania

Odpowiednio zmodyfikowany system GNU/Linux zainstalowany na dedykowanej platformie sprzętowej jest często stosowanym rozwiązaniem w układach sterowania. Dotychczas jednak istnieje niewiele tego typu implementacji wykorzystanych w sterowaniu robotami mobilnymi. Stosunkowo niewielkie rozmiary układu pozwalają na wygodne wykorzystanie go w robotach autonomicznych. Możliwość uruchamiania procesów z rygorem czasu rzeczywistego pozwala na implementację algorytmów sterowania serwomechanizmami odpowiedzialnymi za ruch. Duże możliwości obliczeniowe układu pozwalają natomiast na implementację zaawansowanych algorytmów planowania trajektorii i zastosowanie zaawansowanych technik programistycznych. Łatwość implementacji ułatwia szybkie prototypownie nowych rozwiązań. Otwartość systemu pozwala na łatwe podłączanie nowych urządzeń własnego projektu w szczególności systemów pomiarowych i układów wykonawczych. Obsługa protokołu TCP/IP oraz popularnych kart sieciowych do komunikacji bezprzewodowej (WiFi) ułatwi komunikację z systemem poprzez sieć internet.

Literatura

[1] Abbott D., *Linux for Embedded and Real-Time Applications.* Elsevier Science, 2003.
[2] Barr M., Massa A., *Programming Embedded Systems.* O'Reilly, 2006.
[3] Catsoulis J., *Designing Embedded Hardware*. O'Reilly, 2003.
[4] Corbet J., Rubini A., Kroah-Hartman G., *Linux Device Drivers*. Third Edition, O'Reilly, 2005.
[5] Douglass B., *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison Wesley, 2002.
[6] Piątek M., *Real-Time Application Interface and Xenomai modified GNU/Linux real-time operating systems dedicated to control*. Computer Methods and Systems, Kraków, Akademia Górniczo-Hutnicza 2006.
[7] Pisarczyk P., Jurkiewicz R., Sadowski M., *Systemy wbudowane – kompedium cz. 1 i 2*. Elektronik, 4/2003, 32–35, 5/2003, 38–44.
[8] Yaghmour K., *Buidling Embedded Linux Systems*. O'Reilly, 2003.

[9] http://mars.jpl.nasa.gov/MPF/rover/sojourner.html.
[10] http://www.roomba.pl/.
[11] http://www.kernel.org.
[12] http://www.rtai.org.
[13] http://www.xenomai.org.