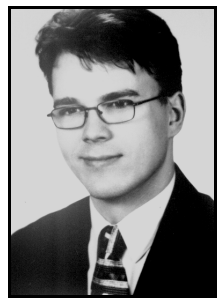
Dariusz ELJASZ, Piotr POWROŹNIK
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT METROLOGII ELEKTRYCZNEJ

Systemy operacyjne w sieciach czujników

Mgr inż. Dariusz ELJASZ



Urodzony w 1979r. Ukończył studia na kierunku Przemysłowe Systemy Informatyczne na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego (2004). Od 2004 roku asystent w Instytucie Metrologii Elektrycznej na macierzystym wydziale. Zainteresowania naukowe określone są w dziedzinie komunikacji bezprzewodowej.

e-mail: D.Eljasz@ime.uz.zgora.pl

Mgr inż. Piotr POWROŹNIK

Ukończył studia na kierunku Przemysłowe Systemy Informatyczne na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego (2004). Od 2004 roku asystent w Instytucie Metrologii Elektrycznej. Zainteresowania naukowe określone są w dziedzinie budowy i oprogramowania systemów kontrolno-pomiarowych w nowych pojawiających się rozwiązaniach technologicznych.

e-mail: P.Powroznik@ime.uz.zgora.pl

Streszczenie

W artykule dokonano porównania wybranych cech dedykowanych systemów operacyjnych TinyOS, SOS oraz MANTIS stosowanych w sieciach czujników. Przedstawiony został sposób funkcjonowania poszczególnych systemów. Zwrócono uwagę na sposób implementacji programowej komunikacji oraz scharakteryzowano dodatkowe narzędzia dostępne z systemem. Ponadto zostały przedstawione spostrzeżenia autorów na temat pracy omawianych systemów z węzłami czujników MICA2.

Słowa kluczowe: sieci czujników, dedykowane systemy operacyjne, TinyOS, SOS, MANTIS.

Operating systems in sensor networks

Abstract

In this paper, selected features for dedicated operating system in sensor networks for systems like TinyOS, SOS and MANTIS are presented. Functions of systems and software implementation for wireless and wired communication is described. Further additional tools available in operating systems for sensor networks are outlined. At the end, remarks on using systems for MICA2 sensors are discussed.

Keywords: sensor networks, embedded operating systems, TinyOS, SOS, MANTIS.

1. Wprowadzenie

Postęp w ostatnich latach umożliwia zwiększenie wydajności mikroprocesorów przy zmniejszeniu zapotrzebowania na energię. Ponadto niski koszt i masowość układów stosowanych w komunikacji bezprzewodowej umożliwia tworzenie sieci czujników.

Obecnie węzły czujników posiadają nie tylko program, ale także system operacyjny do obsługi całego urządzenia. Węzły czujników jako źródła informacji przesyłają dane poprzez sieć czujników. Źródłem danych jest często środowisko, w którym węzeł czujnika się znajduje. Zbierane są takie informacje jak natężenie światła, poziom dźwięku, wilgotność, ciśnienie, temperatura otoczenia lub inne wielkości.

Systemy operacyjne dla węzłów czujników są zwykle prostymi rozwiązaniami. Prostota rozwiązania wynika z faktu, iż cały węzeł czujnika jest urządzeniem o zredukowanych zasobach. System operacyjny pomimo ograniczeń sprzętowych musi posiadać kilka cech istotnych z punktu widzenia przepustowości sieci czujników.

Porównanie wybranych cech dla dedykowanych systemów operacyjnych stosowanych w sieciach czujników zostanie przedstawione dla systemu TinyOS, SOS oraz MANTIS.

2. Charakterystyka systemów dla sieci czujników

Każdy z prezentowanych systemów operacyjnych powstał oraz jest rozwijany w ośrodkach akademickich. Celem tych prac było głównie opracowanie kompleksowych platform do obsługi sieci czujników. Niektóre firmy na bazie określonego systemu zaproponowały własne rozwiązania. Jedną z takich firm jest Meshnetics, która proponuje swoim klientom rozszerzenie systemu TinyOS i języka nesC w postaci meshC [1].

TinyOS jest systemem typu open-source, w którym biblioteki, aplikacje oraz sam system został zaimplementowany w języku strukturalnym opartym o komponenty [2]. Językiem tym jest nesC.

Sam język nesC powstał na potrzeby tworzenia oprogramowania dla dedykowanych systemów takich jak sieci czujników. Podejście takie zostało wymuszone niewielkimi zasobami sprzętowymi, wymagającymi przede wszystkim racjonalnego zarządzenia dostępną pamięcią.

Składnia języka nesC zapewnia wsparcie dla modelu obsługi kilku procesów w tym samym czasie przy jednoczesnej możliwości wymiany informacji pomiędzy procesami. Praktyczna realizacja wymiany informacji w nesC dokonywana jest pomiędzy komponentami.

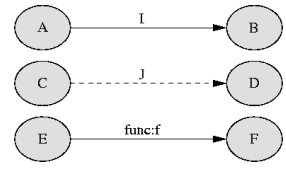
Model obsługi kilku procesów stosowany w nesC oparty jest na zadaniach, zdarzeniach sprzętowych oraz na wykrywaniu w momencie kompilacji możliwości jednoczesnego dostępu lub też zmian określonych zasobów programowych „*data races*".

Aplikacje w nesC składają się z jednego lub więcej komponentów połączonych ze sobą. Łączenie komponentów, określane jest również pojęciem wiązania. Wiązania są dokonywane poprzez stosowanie interfejsów.

Możliwość tworzenia komponentów wraz z dwukierunkowymi interfejsami stanowi ważną funkcjonalność. Dwukierunkowość interfejsów została uzyskana poprzez deklarację zbioru funkcji zwanych „*commands*". Zbiór tych funkcji musi zostać zapewniony przez dostawcę interfejsu. Natomiast inny zbiór funkcji oznaczonych jako „*events*" wymaga już od interfejsu komponentu użytkownika implementacji.

Zaletą języka nesC jest to, że każdy komponent może używać i udostępniać wiele interfejsów oraz wiele instancji pojedynczego interfejsu.

Komponent, który udostępnia i używa interfejsy może zostać powiązany trzema typami wiązań (rys. 1).



Rys. 1. Wiązania komponentów w nesC[1]
Fig. 1. Component wiring in nesC

Na podstawie oznaczeń z rys. 1 definiuje się wiązanie, w którym komponent A wymaga interfejsu I. Interfejs I jest dostarczany

[1] Rysunek wygenerowany programem nesdoc

przez komponent B [2]. Komponenty A i B są ze sobą powiązane. Komponent C i D wymagają natomiast lub dostarczają interfejsu J. Kierunek strzałki wskazuje oryginalne wiązanie C=D. Dla komponentu E wymagana jest funkcja f, która jest dostarczana przez komponent F.

Ponadto nesC zakłada podział komponentów na moduły i konfiguracje. Zadaniem modułu jest dostarczenie kodu aplikacji, która będzie wykonywana na urządzeniu oraz implementacja jednego albo kilku interfejsów. Konfiguracje natomiast dokonują łączenia wszystkich użytych komponentów w aplikacji wraz z użytymi interfejsami.

TinyOS nie został zaprojektowany jako system czasu rzeczywistego. Obsługa nieprzewidywalnej natury komunikacji bezprzewodowej została jednak dotrzymana poprzez w tym celu stworzony model. Zastosowany model obsługi kilku procesów w tym samym czasie przy jednoczesnej możliwości wymiany informacji pomiędzy procesami oparto na zadaniach oraz na zdarzeniach sprzętowych. Cechą charakterystyczną przy wykonywaniu zadań jest brak możliwości wywłaszczenia. Natomiast zdarzenia sprzętowe mają możliwość wywłaszczania zadań, a także innych wywołanych zdarzeń sprzętowych.

W języku nesC przewidziano możliwość umieszczenia w kodzie programu dyrektywy „*atomic*". Dzięki tej dyrektywie możliwe staje się oznaczenie części programu, która musi zostać wykonana w całości. Dopiero po wykonaniu oznaczonego fragmentu kodu będzie możliwe wywłaszczenie.

Tak jak w przypadku TinyOS, system operacyjny SOS dedykowany jest dla bezprzewodowych sieci czujników. System ten również nie jest systemem czasu rzeczywistego. SOS oparty został na modułach [4]. Samo jądro systemu jest również modułem potrafiącym dynamicznie zarządzać pamięcią (zwalniać i przydzielać zasoby), dynamicznie dołączać i odłączać moduły programu. Dynamiczne ładowanie modułów programowych do pamięci daje możliwość obsługi węzłów czujników oraz zapewnia wsparcie dla usług sieciowych. Ponadto system ma wbudowane proste mechanizmy zabezpieczające przed przeciążeniem pamięci (*Guard bytes for run-time memory over-flow checks*). Wszystkie komunikaty trafiają do jądra systemu w wyniku czego są przez nie obsługiwane. Jak zapewniają twórcy, system SOS działa nie gorzej niż TinyOS, który jest systemem dedykowanym dla węzłów czujników pracujących bezprzewodowo. Nie zajmuje znacząco więcej miejsca w pamięci operacyjnej urządzeń oraz jest porównywalnie szybki w działaniu. Wszystkie funkcjonalności systemu SOS zostały zaimplementowane w języku C. W tym przypadku nie podjęto próby tworzenia nowego standardu dla języka programowania.

Podobne podejście przyjęto w przypadku systemu MANTIS (MultimodAl NeTworks of In-situ Sensors) [5]. Wszystkie składowe tego systemu zostały zaimplementowane w języku C zapewniając programowe oraz sprzętowe wsparcie dla określonych platform sieci czujników. Wśród najważniejszych cech tego systemu wymienia się wielowątkowość oraz automatyczne wywłaszczanie procesów, w przypadku, gdy upłynął określony przedział czasowy. Czas przełączania kontekstu określono na około 200 μs. MANTIS nie wymaga dużych zasobów pamięciowych przeznaczonych na sam system operacyjny. Twórcy podają iż jest to 500B pamięci RAM oraz 14kB pamięci flash. Niewielkie wymagania na pamięć systemu operacyjnego umożliwiają budowanie bardziej zaawansowanych aplikacji. W systemie MANTIS zaimplementowane jest także wywłaszczanie wielowątkowe. Zarządca wątków został oparty na algorytmie obsługi *round-robin* wraz z możliwością nadawania priorytetów. Wprowadzono dwa poziomy priorytetów. MANTIS tak jak dwa poprzednie systemy zapewnia wsparcie dla odczytu danych z węzła czujnika oraz ich przesłania, gdy zachodzi taka potrzeba do dalszego przetworzenia.

Cechą, którą należy wziąć pod uwagą w przypadku posiadania określonej platformy sprzętowej sieci czujników jest wsparcie programowe określonego systemu w postaci dołączonych bibliotek np. do obsługi określonego typu mikroprocesora lub też modułu radiowego. W tab. 1 zestawiono wsparcie omawianych systemów w artykule.

Tab. 1. Obsługiwane platformy sprzętowe dla sieci czujników
Tab. 1. Support for hardware platform in sensor networks

TinyOS[2]	*SOS*[3]	*MANTIS*
EYESIFXV2	MICA2	MICA2
INTELMOTE2	CYCLOPS	MICA2DOT
MICA2	XYZ	MICAZ
MICA2DOT		TELOS rev. B
MICAZ		x86 Linux
TELOSB		
TINYNODE		
BTNODE3		

Na podstawie tab. 1 można zauważyć, że wszystkie opisywane systemy wspierają platformy oparte na mikroprocesorze ATmega128L firmy Atmel oraz układ radiowy CC1000 lub CC2420 firmy Chipcon.

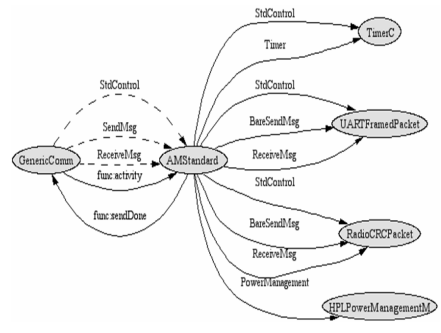
3. Implementacja programowa komunikacji bezprzewodowej oraz przewodowej pomiędzy węzłami czujników działającymi na określonym systemie operacyjnym

Przesyłane dane w systemach operacyjnych dla sieci czujników mogą być na bieżąco przetwarzane lub też gromadzone. Przetwarzaniu może towarzyszyć tworzenie reguł. Na podstawie reguł staje się możliwe sterowanie pojedynczymi urządzeniami lub też grupami urządzeń.

Dane mogą być przesyłane drogą radiową wewnątrz sieci czujników lub przesyłane na zewnątrz sieci czujników poprzez odpowiednie bramy (*Gateway*).

W systemie TinyOS przesyłanie danych do bramy jest możliwe po zastosowaniu odpowiednich komponentów. Cechą wspólną tego rodzaju komponentów jest utworzenie określonej struktury ramki. Zawartość ramki jest następnie wysyłana w wyniku wywołania określonego interfejsu będącego funkcją typu „commands". Moment wysłania informacji jest sygnalizowany również poprzez interfejs, który jest już funkcją typu „events".

Komponentem podstawowym, umożliwiającym przesyłanie informacji w systemie TinyOS jest komponent GenericComm (rys. 2).



Rys. 2. Komponenty odpowiedzialne za komunikację w TinyOS[4]
Fig. 2. Components responsible for communications in TinyOS operating system

Na bazie komponentu GenericComm w wyniku dowiązań powstały rozszerzenia dodające kolejne funkcjonalności umożliwiające lepsze zarządzanie siecią czujników.

[2] Wspierane platformy sprzętowe dla TinyOS w wersji 2.0
[3] W Tab. 1. przedstawiono platformy, dla których system SOS udziela pełnego wsparcia oraz został w pełni przetestowany
[4] Rysunek wygenerowany programem nesdoc

Komponent GenericComm zapewnia nie tylko możliwość wysyłania danych, ale również odbierania (rys. 2). Odbieranie następuje w wyniku wywołania interfejsu będącego funkcją typu „events": ReceiveMsg. W wyniku wywołania wcześniej wspomnianego interfejsu użytkownik otrzymuje wskaźnik do struktury, w której znajdują się dane.

System TinyOS umożliwia zdefiniowanie rozmiaru przesyłanych danych. W przypadku, gdy rozmiar przesyłanych danych będzie większą wartością, niż maksymalny rozmiar danych wynikający ze standardu komunikacyjnego IEEE 802.15.4, następuje automatyczna programowa fragmentacja danych.

Jak już wcześniej napisano system operacyjny MANTIS bazuje na tradycyjnym języku C. W związku z tym twórcy systemu napisali funkcje odpowiedzialne za wysyłanie oraz odbieranie danych. Chcąc wysłać dane użytkownik musi jedynie wywołać funkcję za to odpowiedzialną. Com_send(iface,buf) jest funkcją, w której „iface" jest interfejsem (radio lub RS-232), do którego wysyłane są dane zgromadzone w zmiennej „buf". Aby odebrać dane należy wywołać funkcję com_recv(iface), która zwraca dane znajdujące się w interfejsie „iface". W wyniku takiej operacji, podobnie jak ma to miejsce w systemie TinyOS, użytkownik otrzymuje wskaźnik do struktury, w której znajdują się dane. Implementacja w/w funkcji znajduje się w bibliotece COM.H.

Twórcy systemu SOS przyjęli nieco odmienną koncepcję tworzenia oprogramowania. Stworzyli oni system mający niektóre cechy TinyOS-a lecz napisanego w języku C. W systemie tym poszczególne funkcje znajdują się w modułach. Użytkownik ma do dyspozycji 9 modułów:

- Memory Allocation,
- Timer,
- Messaging,
- Node Info,
- Random Numbers,
- System Time,
- Sensing,
- LED functions,
- Module State.

Ma to na celu zaoszczędzenie pamięci potrzebnej dla aplikacji. SOS posiada mechanizm dynamicznego podłączania i odłączania poszczególnych modułów, co w znaczący sposób może zmniejszyć kod programu. Funkcje związane z wysyłaniem i odbieraniem danych znajdują się w module „*Messaging*", funkcją za to odpowiedzialną jest sys_post(dst_mod_id, type, size, *data, flag), w której jako argumenty podaje się gdzie ma informacja być wysłana, jaki ma rozmiar oraz jaki zostanie nadany dla niej priorytet (zmienna *flag*). Chcąc wysłać/odebrać dane do/z RS-232/Ethernetu należy wywołać funkcję sys_post_UART/ sys_post_NET. Implementacja funkcji opisanych powyżej znajduje się w bibliotece SYS_MODULE.H.

Odporność na błędy podczas pracy oraz szyfrowania danych ma kluczowe znaczenie dla sieci czujników stosowanych w miejscach, gdzie mają bezpośredni wpływ na życie człowieka.

Dla systemu TinyOS został zaimplementowany komponent TinySec[6] umożliwiający szyfrowanie przesyłanych danych. Poziom bezpieczeństwa określany jest na dwóch poziomach. Pierwszy poziom uwierzytelnia polega na dodaniu do przesyłanych danych uwierzytelniającego kodu wiadomości Message Authentication Codes (MACs). Kod ten jest generowany przez nadawcę na podstawie sumy kontrolnej przesyłanej wiadomości oraz zdefiniowanego wcześniej klucza. Odbiorca posiada także ten sam klucz, przez co może obliczyć MAC danych z wiadomości, która do niego dotarła. Jeżeli MAC wygenerowany przez odbiorcę zgadza się z wartością MAC znajdującą się w przesłanej wiadomości wówczas wiadomość jest przyjmowana. Możliwe jest również załączenie drugiego poziomu. W tym przypadku przed obliczeniem wartości MAC i wysłaniem wiadomości szyfrowane są dodatkowo dane przez nadawcę. Proces uwierzytelnia danych przez odbiorcę jest podobny jak w przypadku pierwszym, z wyjątkiem tego, że już po uwierzytelnieniu wiadomości dane muszą być dodatkowo rozszyfrowane.

4. Narzędzia dostępne z systemami

Do pełnego wykorzystania możliwości systemu niezbędne jest posiadanie odpowiednich narzędzi programowych dla zdefiniowanego języka programowania. Narzędzia takie wspomagają proces tworzenia aplikacji. Mogą również znacząco wspomóc rozwiązywanie problemów związanych z błędnie funkcjonującym oprogramowaniem lub też z uszkodzonym sprzętem. Umożliwiają ponadto dwukierunkową komunikację sieci czujników najczęściej z aplikacjami PC.

Do systemu TinyOS zaimplementowano kilka bardzo przydatnych narzędzi. Narzędzia te zostały utworzone najczęściej w środowisku JAVA. Twórcy narzędzi udostępniają również kod źródłowy do swojego oprogramowania, dzięki temu często staje się możliwe poznanie sposobu funkcjonowania określonych bloków funkcjonalnych sprzętu.

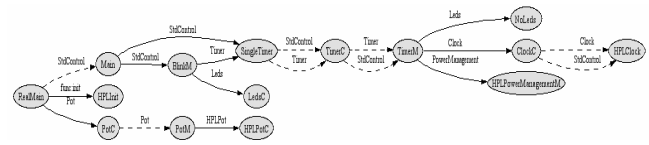
Aplikacja SerialForwarder umożliwia przesyłanie danych w obie strony pomiędzy siecią czujnikową a komputerem PC przy wykorzystaniu bramy (*Gateway*).

Formatowaniem danych do postaci tekstowej pochodzących z sieci czujnikowej zajmuje się program xlisten. Program ten stanowi uzupełnienie do aplikacji testowych przeznaczonych dla węzłów czujników.

Graficzną formę sieci czujnikowej uzyskuje się po uruchomieniu aplikacji Sensor Network Topology lub też Mote View.

Również dla środowiska Matlab zostały dodane odpowiednie sterowniki, dzięki, którym można przeprowadzać bardziej zaawansowane analizy samej sieci czujników lub też uzyskiwanych danych.

Niezwykle pomocnym narzędziem w procesie tworzenia oprogramowania dla węzłów czujników jest program nescdoc. W rezultacie uruchomienia tego programu otrzymywana jest na podstawie kodu źródłowego w nesC dokumentacja. Na podstawie dokumentacji dużo łatwiej jest sprawdzić wiązania pomiędzy użytymi komponentami (rys. 3).



Rys. 3. Wiązania komponentów dla przykładowej aplikacji[5]
Fig. 3. Component wiring for selected application

Do graficznego debuggowania programów przewidziano program TinyViz.

Przy tworzeniu własnego oprogramowania użytkowego na PC dla wizualizacji stanu sieci czujnikowej jest użyteczny program Message Interface Generator dla języka nesC. Program ten na podstawie kodu źródłowego przeznaczonego dla węzłów czujników generuje plik w języku JAVA. Wygenerowany kod zapewnia prawidłowy format przy przesyłanych i odbieranych danych z sieci czujnikowej do PC.

System SOS umożliwia tak samo jak TinyOS przeprowadzenie symulacji działania napisanego programu, bez konieczności ładowania go do urządzenia. Oprócz symulatora dostępnego wraz z system SOS, programy można również symulować narzędziami powstałymi w wyniku prac związanych z projektem Avrora. Narzędzia zostały zaprojektowane do symulowania przede wszystkim mikrokontrolerów AVR produkowanych przez firmę Atmel oraz węzłów czujników MICA2 firmy Crossbow.

System MANTIS jest najbardziej ubogi spośród wymienionych systemów operacyjnych w dodatkowy zestaw użytecznych narzędzi. Prawdopodobnie wynika to z faktu, że jest to rozwiązanie czysto akademickie, a dodatkowo cały system jest napisany w języku C. Warto jednak zwrócić uwagę, że MANTIS posiada najlepiej spośród w/w systemów opracowany system pomocy i opisu działania funkcji.

[5] Rysunek wygenerowany programem nesdoc

5. Spostrzeżenia dotyczące obsługi systemów na urządzeniach MICA2

Przedstawione wnioski i spostrzeżenia opisane w niniejszym rozdziale powstały na podstawie doświadczeń związanych z tworzeniem oprogramowania przez autorów dla węzłów czujników MICA2 firmy Crossbow.

Kompilacja programów dla każdego z przedstawianych systemów dla sieci czujnikowych jest możliwa w środowiskach UNIXowych. Środowisko Cygwin daje możliwość kompilacji w systemach Windows.

W systemie TinyOS podstawowym kompilatorem jest ncc. Jest to odmiana darmowego kompilatora gcc przeznaczona do współpracy z językiem programowania nesC. Z uwagi na sposób linkowania bibliotek, proces kompilacji przebiega bez większych problemów. Kompilator automatycznie pobiera wszelkie potrzebne biblioteki i źródła. Po udanej kompilacji na ekranie monitora ukazuje się ile program będzie zajmował miejsca w pamięci RAM oraz flash. Wraz z napisaną aplikacją do kodu źródłowego dołączany jest bootloader umożliwiający załadowanie programu do urządzenia. Instalacja samego systemu operacyjnego TinyOS, także nie przysparza kłopotu. Wszelkie potrzebne biblioteki wraz z dodatkowym oprogramowaniem są dostępne na stronie projektu [3].

Jako kompilator dla systemu SOS, można użyć każdego kompilatora języka C. Standardowo dodawany jest darmowy kompilator AVR-a, gcc. Programy można debaggować poprzez GDB. JTAG nie jest w pełni obsługiwany. Aby skompilować system, najpierw trzeba ściągnąć źródła systemu (aktualnie jest to wersja 2.0). Następnie wymagane jest pobranie dodatkowych pakietów (Cross-compiling), dzięki którym program napisany na komputerze PC będzie mógł być uruchomiony na węźle czujnika (np. MICA2). Dodatkowo wymagany jest także kompilator oraz biblioteki języka C. System SOS obsługuje procesory firmy AVR oraz MSP430. Każda z platform także posiada własny Cross-compiler. W celu uzyskania kodu dla węzła czujnika dla systemu SOS najpierw należy skompilować jądro systemu (BLANK). Kolejnym krokiem jest uruchomienie serwera x86 pośredniczącego w wymianie informacji pomiędzy komputerem PC a urządzeniem, oraz z uwagi na modułową budowę systemu (dynamiczna możliwość dołączani i odłączania poszczególnych modułów programu) uruchomić narzędzie zarządzające modułami. Po wykonaniu tych wszystkich czynności można załadować oprogramowanie wraz z jądrem do węzła czujnika.

Podobnie jak w SOS-ie, w systemie MANTIS wykorzystywane jest darmowy kompilator gcc. Kompilacja oprogramowania jest wyraźnie dłuższa niż w przypadku TinyOS. Różnica czasu wynika z faktu, iż dołączane są poszczególne biblioteki, a nie tylko to co jest potrzebne.

Przed przystąpieniem do wgrywania oprogramowania należy do węzła czujnika załadować bootloader. Robi się to tylko raz. Można także wgrywać oprogramowanie przekompilowane przez TiniOS.

Z uwagi na zastosowanie różnych filozofii tworzenia oprogramowania, każdy z w/w systemów cechuje się różnym zapotrzebowaniem na pamięć. Przykładowy rozmiar aplikacji pozwalającej na zapalenie diod znajdujących się na modułach MICA2 przedstawiono w tabeli 2.

Tab. 2. Wielkość kodu dla prostej aplikacji
Tab. 2. Code size for simple application

System operacyjny	Wielkość kodu [B]
TinyOS	1 859
SOS	740 + 102 848 jądro
MANTIS	52 852

Z danych przedstawionych w tab. 2 wynika, że w przypadku prostej aplikacji najmniejsze zapotrzebowanie na pamięć posiada system TinyOS.

6. Podsumowanie

W artykule podjęto próbę przedstawienia wybranych cech dotyczących systemów operacyjnych dla sieci czujników na przykładzie systemu TinyOS, SOS oraz MANTIS.

Cechą wspólną systemów dla sieci czujników jest odczytywanie danych, a następnie przetworzenie lokalne lub też przesłanie uzyskanych rezultatów. Przesłanie danych odbywa się wewnątrz sieci czujników zazwyczaj w formie bezprzewodowej lub też po zastosowaniu bram dane mogą być dostępne na zewnątrz sieci czujnikowej.

Przy przesyłaniu danych uwzględnia się konieczność autoryzacji źródła oraz jeżeli zachodzi taka potrzeba szyfrowania danych. Stosowanie dodatkowych zabezpieczeń jest stosowane w przypadku, gdy istnieje niebezpieczeństwo podejmowania prób włamań do sieci czujników. W wyniku takiego włamania istnieje możliwość nieautoryzowanego dostępu do przesyłanych danych lub też może zajść próba destabilizacji pracy działających węzłów czujników w sieci.

Na potrzeby określonego systemu opracowywany jest również dedykowany styl tworzenia oprogramowania. Dzięki takiemu podejściu staje się możliwa obsługa węzła czujnika, który posiada ograniczoną ilość dostępnych zasobów. Zastosowanie komponentów w TinyOS lub modułów w SOS umożliwia zmniejszenie generowanego kodu aplikacji. Możliwe staję się również lepsze zarządzanie ograniczonej ilości pamięci.

W przypadku obsługi komponentów w systemie TinyOS zdecydowano się na utworzenie nowego języka programowania w postaci nesC będącego rozszerzeniem języka C. Pozostałe dwa systemy wykorzystają do implementacji oprogramowania dla węzłów czujników języka C.

Pod względem dostępnego oprogramowania wraz z systemem najlepiej wypada system TinyOS. Jednak dla pozostałych dwóch systemów możliwe jest często też skorzystanie z narzędzi TinyOS.

W przypadku wszystkich systemów do lepszego wykorzystania oferowanych możliwości przez te systemy możliwe jest również skorzystanie z dodatkowych programów nie dostarczanych razem z systemem.

Na długość nieprzerwanej pracy systemu operacyjnego wpływ ma poziom zużycia energii przez węzeł czujnika. Z tego powodu w systemie operacyjnym muszą zostać zaimplementowane funkcje umożliwiające redukcję zużycia energii dla urządzenia pracującego w trybie czuwania. W artykule nie został rozważony ten aspekt ze względu na szeroką tematykę tego zagadnienia.

7. Literatura

[1] A. Belenki: Overcoming Challenges of TinyOS Use in Commercial ZigBee Applications. III International Technology Exchange, Stanford University, February 10, 2006.

[2] W. Hong, S. MAdden: TinySchema: Managing Attributes, Commands and Events in TinyOS, September, 2003.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister: System Architecture Directions for Networked Sensors. Department of Electrical Engineering and Computer Sciences University of California, Berkeley, CA.

[4] Chih-Chieh Han, Ram Kumar Rengaswamy, Roy Shea, Eddie Kohler and Mani Srivastava: A Dynamic Operating System for Sensor Nodes. Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys), 2005.

[5] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, Richard Han: MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. ACMKluwer Mobile Networks & Applications (MONET) Journal, Special Issue on Wireless Sensor Networks, August 2005.

[6] C. Karlof, N. Sastry, D. Wagner: TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. Baltimore, Maryland, USA, SenSys'04, November 3–5, 2004.

Artykul recenzowany