Przegląd systemów operacyjnych dla systemów wbudowanych

Użytkownicy rozpieszczani kolorowymi wyświetlaczami oraz interfejsami dotykowymi telefonów komórkowych stawiają przed konstruktorami urządzeń elektronicznych coraz większe wymagania. Niegdyś popularne monochromatyczne wyświetlacze znakowe odchodzą do lamusa nawet w urządzeniach amatorskich. Równocześnie czasy prostych interfejsów komunikacyjnych, takich jak np. RS232, którego oprogramowanie można było zmieścić w kilkudziesięciu liniach kodu, również odeszły w zapomnienie. Współczesne interfejsy takie jak USB, Ethernet, CAN, Wi-Fi, Bluetooth, wymagają skomplikowanych rozwiązań programowych, zarówno do obsługi samych rejestrów sprzętowych kontrolera interfejsu, jak i dodatkowych warstw abstrakcji. Wszystko to powoduje, że przygotowanie we własnym zakresie odpowiedniego oprogramowania od podstaw dla nieco bardziej zaawansowanych projektów, za pomocą samego kompilatora, jest zadaniem żmudnym, podatnym na błędy, a co za tym idzie kosztownym. Rozwiązaniem jest zastosowanie systemu operacyjnego.

Tworzenie całego oprogramowania od podstaw ma czasem sens (choć nie zawsze) jedynie w dużych firmach dla produkcji wielkoseryjnej, gdzie pracochłonny koszt przygotowania wyspecjalizowanego oprogramowania umożliwia zastosowanie prostszych, a zatem tańszych podzespołów. Dlatego w urządzeniach elektronicznych coraz chętniej stosowane są systemy operacyjne, które doskonale upraszczają proces przygotowania oprogramowania pozwalając skupić się na funkcjonalności, dodając kolejną warstwę abstrakcji pomiędzy programem a sprzętem. Podobnie jak kilkanaście lat temu w oprogramowaniu wbudowanym odeszliśmy od pisania kodu aplikacji w assemblerze (z wyjątkiem nielicznych wstawek) na rzecz języka C, który nazywany jest „przenośnym assemblerem", tak dzisiaj obserwujemy proces odchodzenia od bezpośredniego programowania rejestrów mikrokontrolera na rzecz systemów operacyjnych oraz bibliotek dodatkowych. Nie bez znaczenia jednym z dodatkowych powodów coraz większej popularności takich rozwiązań jest rewolucja, która dokonała się w przeciągu ostatnich kilku, lat polegająca na odchodzeniu od mikrokontrolerów 8-bitowych w stronę rozwiązań 32-bitowych (głównie ARM). Współczesne mikrokontrolery mają wielokrotnie bardziej wydajną jednostkę centralną oraz zasoby pamięci. Umożliwiają przy tym uruchamianie systemów operacyjnych bez zauważalnej utraty wydajności. Często wykorzystanie systemów operacyjnych nie wymaga ponoszenia dodatkowych kosztów, ponieważ wiele z nich jest dostępnych na wolnych licencji umożliwiającej pobranie i modyfikowanie źródła, przy czym gro z nich to projekty dojrzałe, mające certyfikaty lub umożliwiające wykupienie wsparcia komercyjnego. Istnieją również komercyjne systemy o zamkniętym kodzie źródłowym, których użycie wymaga poniesienia dodatkowych kosztów zakupu samej licencji. W zależności od sposobu licencjonowania, może to być opłata jednorazowa lub też opłata od każdej sztuki działającego urządzenia.

Architektura

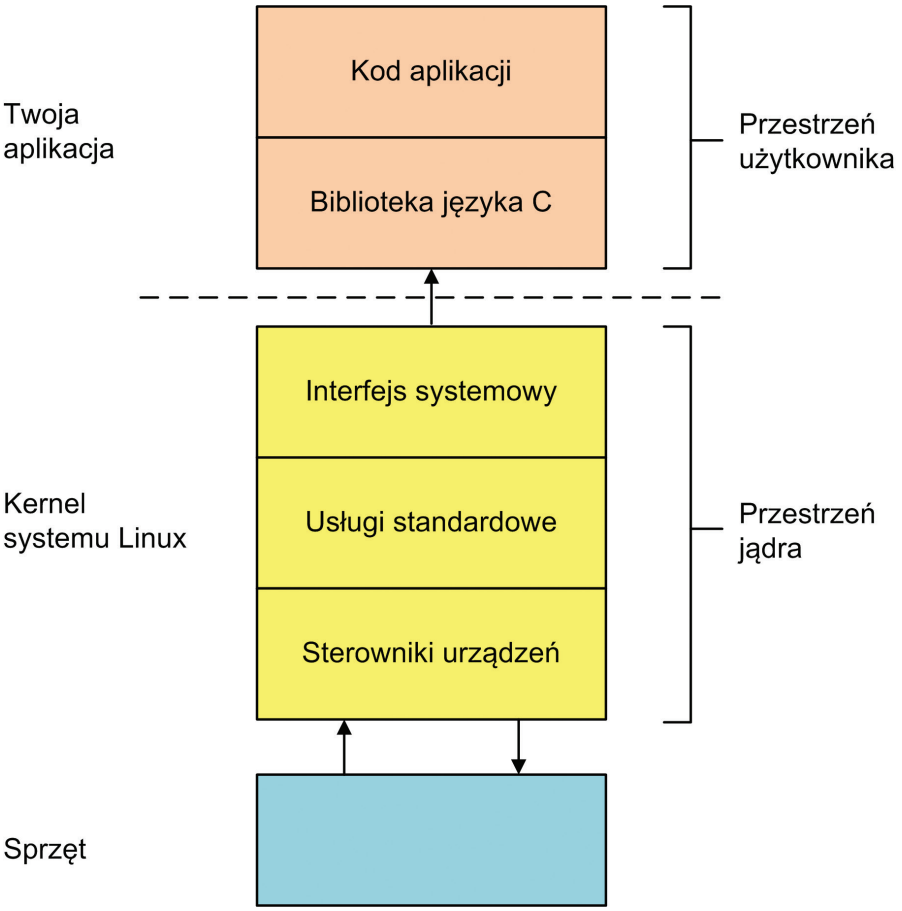
Pod pojęciem systemu operacyjnego dla urządzeń wbudowanych kryje się cała gama systemów, od tych najprostszych, które możemy uruchomić na mikrokontrolerach 8-bitowych, po rozbudowane, znane z komputerów PC. Ze względu na rodzaj obsługiwanych układów systemy możemy podzielić na dwie kategorie:

- Systemy operacyjne przeznaczone dla mikroprocesorów minimum 32-bitowych wyposażonych w jednostkę zarządzania pamięcią MMU oraz dużą, zewnętrzną pamięć operacyjną, zazwyczaj większą niż 4 MB.
- Minisystemy operacyjne, przeznaczone dla mikrokontrolerów jednoukładowych, pracujące w pojedynczej przestrzeni adresowej i wymagające niewielkiej pamięci operacyjnej rzędu pojedynczych kB.

Systemy z pierwszej grupy, są najczęściej rozwiązaniami zaawansowanymi, zapewniającymi sterowniki dla wielu układów peryferyjnych, wbudowaną obsługę zaawansowanych protokołów sieciowych, wyświetlaczy graficznych, klawiatur itp. Wspierają one również separację oraz wirtualizację zasobów pomiędzy poszczególnymi aplikacjami, dzięki czemu nawet w przypadku awarii jednej z aplikacji, pozostała część systemu oraz aplikacja nadal pracują prawidłowo. Typowym przedstawicielem systemu wymagającego mikroprocesora z MMU jest Linux, którego architekturę programową przedstawiono poglądowo na **rysunku 1**.

Linux składa się z monolitycznego jądra odpowiedzialnego za wirtualizację zasobów, szeregowanie zadań, komunikację międzyprocesową, obsługę protokołów sieciowych. Jądro zapewnia także jednolity interfejs dostępu do urządzeń dodając dodatkową warstwę abstrakcji uniezależniającą aplikację od sprzętu, na którym pracuje. Bezpośredni dostęp do sprzętu ma jedynie jądro systemu operacyjnego. Wszystkie aplikacje korzystające ze sprzętu muszą odwoływać się do niego za pośrednictwem jądra. Jądro pracuje w trybie uprzywilejowanym, mając dostęp do całej przestrzeni adresowej oraz urządzeń peryferyjnych. Aplikacje działają w przestrzeni użytkownika i mają ograniczony dostęp do zasobów systemowych pracując w wydzielonych, niezależnych przestrzeniach adresowych. Dostęp do wywołań systemowych jądra odbywa się za pomocą biblioteki *libc*, która zapewnia dodatkową funkcjonalność oraz tworzy warstwę abstrakcji pomiędzy interfejsem jądra a aplikacją użytkownika.

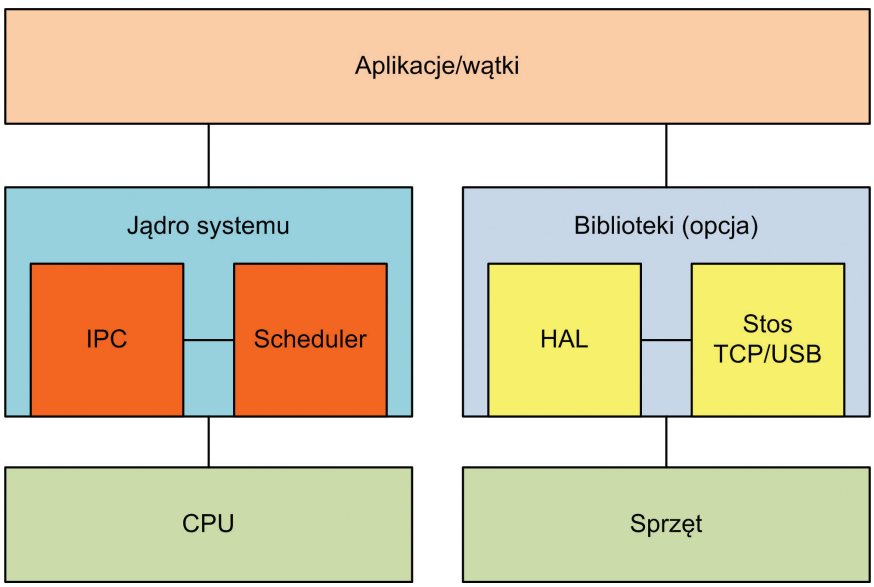
Pomimo niezaprzeczalnych, zalet pełnoprawne systemy operacyjne narzucają stosunkowo duże wymagania odnośnie do



Rysunek 1. Poglądowa architektura systemu Linux embedded

samej jednostki centralnej oraz wymagają dodatkowych zasobów, takich jak stosunkowo duża pamięć operacyjna czy dodatkowa pamięć stała, w której będą przechowywane pliki oraz aplikacje. Niestety, współczesne mikrokontrolery jednoukładowe nie mają zasobów umożliwiających uruchomienie pełnoprawnego systemu. W takich wypadkach znajdują zastosowanie *„minisystemy operacyjne”* o znacznie uproszczonej budowie, oparte o koncepcję mikrojądra. Systemy te mają zdecydowanie mniejsze wymagania. Przede wszystkim nie jest wymagana jednostka centralna wyposażona w MMU, a apetyt na pamięć jest znacznie mniejszy. Do jego uruchomienia wystarczy nawet 8-bitowy mikrokontroler wyposażony w pamięci RAM i Flash o wielkości kilkunastu KB.

Podstawową architekturę takiego minisystemu pokazano na **rysunku 2**. Brak wymagań odnośnie do MMU, powoduje, że wszystkie aplikacje oraz sam system działają we wspólnej przestrzeni adresowej. Aplikacje oraz jądro mają pełny dostęp do siebie nawzajem oraz do wszystkich zasobów sprzętowych. Kod systemu oraz aplikacje linkowane są statycznie do pojedynczego obrazu, który jest następnie wgrywany do pamięci Flash urządzenia. Główną częścią systemu operacyjnego jest jądro, które jest bardzo proste i składa się z modułu szeregowania zadań (*schedulera*) oraz modułu komunikacji międzyprocesowej (*IPC*). Moduł szeregowania zadań jest odpowiedzialny za przydzielanie czasu procesora poszczególnym aplikacjom, natomiast moduł **IPC** jest odpowiedzialny za komunikację pomiędzy procesami oraz jądrem. Aplikacje mają dostęp do wszystkich zasobów, zatem nie jest konieczne odwoływanie się do sprzętu za pośrednictwem jądra. W tej architekturze można odwoływać się bezpośrednio do sprzętu za pomocą wpisów do rejestrów sprzętu, tak jak w przypadku klasycznych aplikacji pozbawionych systemu.



Rysunek 2. Uproszczona architektura minisystemu operacyjnego

Bardzo często minisystemy mają dodatkowe moduły udostępniające warstwę abstrakcji sprzętu HAL czy integrujące np. stos TCP lub USB, niestanowiące w ścisłym sensie jądra systemu, ale dostarczane razem z nim. Dodatkowa warstwa nie jest wymagana, ale pozwala na uniezależnienie aplikacji od specyfiki danego rozwiązania sprzętowego. Często również minisystemy wyposażane są w dodatkowe stosy protokołów np. TCP/IP, USB, Bluetooth itp., zapewniając odpowiednie API. Zazwyczaj istnieje możliwość wyłączenia poszczególnych modułów na etapie kompilowania i pozostawienie minimalnego zestawu składającego się jedynie z jądra. Rozpatrując minisystem z punktu widzenia pełnoprawnego systemu możemy powiedzieć, że jego jądro stanowi analogię do biblioteki obsługi wątków np. Linuksowego *pthread*, a poszczególne aplikacje należy rozpatrywać bardziej w kategoriach wątków, niż procesów z uwagi na pracę we wspólnej przestrzeni adresowej.

Wybór platformy sprzętowej. Mikrokontroler jednoukładowy czy mikroprocesor z MMU?

Do niedawna wybór pomiędzy mikrokontrolerem, a mikroprocesorem z MMU dla średniozaawansowanych projektów był oczywisty. Podstawowym wyborem był mikrokontroler jednoukładowy z uwagi na korzystną cenę lub o wiele prostszy projekt płytki drukowanej. Jedynie w wypadku najbardziej zaawansowanych projektów mogliśmy sobie pozwolić na użycie mikroprocesora oraz pełnego systemu operacyjnego. Dziś porównując cenę oraz możliwości współczesnych układów wybór już nie jest taki oczywisty. Układ umożliwiający uruchomienie kompletnego systemu operacyjnego możemy nabyć w cenie detalicznej około 20 złotych. Uzupełniając go o pamięć DRAM oraz niewielką pamięć Flash, łącznie za około 10

złotych, otrzymujemy kompletny system komputerowy, który będzie tańszy niż np. mikrokontroler jednoukładowy STM32F4 mający przy tym znacznie mniejsze możliwości. Zastosowanie pełnoprawnego systemu operacyjnego niesie ze sobą szereg zalet od strony programowej, nieosiągalnych w wypadku użycia bibliotek dostarczanych przez producenta minisystemu. Najistotniejszą kwestią jest dostęp do ogromnej bazy bibliotek oraz programów, które często dostępne są w postaci gotowych rozwiązań i kodów źródłowych. Wiele popularnych systemów udostępnia sterowniki urządzeń, takich jak pamięci USB, moduły Wi-Fi, Ethernet, Bluetooth oraz kompletne stosy protokołów niedostępne dla małych systemów. Dokonując wyboru musimy zastanowić się czy kosztem większego skomplikowania projektu sprzętowego nie uzyskamy znaczącego obniżenia kosztów przygotowania oprogramowania, które dla projektów nisko- i średniona-kładowych stanowi znaczący koszt. Dzięki zastosowaniu dużej bazy oprogramowania możemy skupić się na tworzeniu aplikacji zamiast kolejny raz „wynajdować koło" i uruchamiać czy pisać np. kod obsługi interfejsu USB.

Również bardzo istotną kwestią jest dostępność MMU oraz odrębnej pamięci wirtualnej, oddzielającej od siebie poszczególne procesy oraz jądro. Rozwiązanie to ułatwia tworzenie zaawansowanego oprogramowania oraz podnosi poziom bezpieczeństwa gotowego rozwiązania. Nietrudno wyobrazić sobie, jakim koszmarem jest szukanie błędów w rozbudowanym oprogramowaniu, gdy nie mamy do dyspozycji mechanizmów ochrony pamięci, gdzie błąd popełniony w jednym fragmencie może ujawnić się w zupełnie innym. Z tych właśnie powodów należy rozważyć czy nie wygodniej i rozsądniej będzie użyć bardziej zaawansowa-

Tabela 1. Najbardziej popularne systemy operacyjne

Nazwa	Platforma	RAM/ FLASH	Priorytety	C++	HAL	Filesys-tem	USB	TCPIP	Producent	Licencja
ChibiOS	I386 ARM7 ARM9 Cortex-M0, M3, M4 PPC e200z AVR8 MSP430 STM8 Coldfire H8S	2K/2-6K	+/RR	+	+	+	+	+	?	GPL3/Commercial
FreeRTOS	ARM Cortex-M3,M4 AVR AVR32 HCS12 MSP430 PIC HC8S x86 8052 Coldfire V850	4K/6K	+/FIXED	-	+	-	-	+		GPL/Commercial Support
ISIX	ARM7/ Cortex M3,M4	2K/4K	+	+	- (częściowo)	+	+	+	BoFF	GPL
MQX	ColdFire PowerPC ARC ARM xSCALE	2K/12K	+	+	+	+	+	+	Freescale	Proprietary
Contiki	MSP430 AVR STM32 PIC32 6502	1K/6K	- (Cooperative scheduler)	-	+	+	+	+	Adam Dunklers and Community	BSD
ECOS	ARM, CalmRISC, FR-V, H8, IA32, MC68000 MIPS, NIOS II, PowerPC SPARC, SuperH	512k/512k	+	+	+	+	+	+	Community	GPL
BeRTOS	Cortex-M3, ARM7TDMI, AVR, x86	kilka KB	+	-	+	+	+(beta)	+	Develever s.r.l. and community	Modified GPLv2
Erika Enterprise	ARM7, ARM9, Cortex-M, PIC24, PIC32, ST10, S12XS, PPC, NiosII, R2xx	1-4KB	+ (multicore)	-	-	-	-	+	GPL linking exception	Evidence Srl ReTiS Lab. and others
FunkOS	AVR, ARM, MSP430	few KB	+	+	+	+	-	-	Sleepycat licence (OpenSource)	Funkenstein Software Consulting
NucleusRTOS	ARM MIPS PPC NiosII MicroBlaze SuperH AT91SAM7	2k/2k	+	+	+	+	+	+	Mentor Graphics	Proprietary
MicroC/OS-II	Cortex-M3 Cortex-M4F ARM7TDMI AVR	few kB/20kB	+	?	+	+	+	+	Micrium	Commercial, free for College and Universities
UCLinux	NIOS, Blackfin ARM7TDMI MC68000 H8 i960 MIPS ColdFire V850 MicroBlaze	4M/4M	+	+	+	+	+	+	Community	GPL

nego sprzętu i pełnoprawnego systemu, zamiast tracić czas na uruchamianie aplikacji dla mikrokontrolera.

Przegląd popularnych mini systemów operacyjnych

Na rynku istnieje wiele różnych systemów operacyjnych przeznaczonych dla mikrokontrolerów, przy czym w przeciwieństwie do systemów wymagających MMU, istnieje duża różnorodność. Aby nie zagubić się w tym gąszczu opiszemy najbardziej popularne systemy, które możemy brać pod uwagę chcąc zastosować je we własnych projektach. Duża część z nich jest dostępna na zasadach otwartych licencji, więc dostępne będą do nich kody źródłowe oraz będziemy mogli ich użyć bez ponoszenia dodatkowych opłat. Jeśli istnieje konieczność wsparcia technicznego, wiele projektów umożliwia wykupienie komercyjnego wsparcia. Dodatkową zaletą przemawiającą za użyciem systemu o otwartej licencji jest ogromna rzesza użytkowników, co pozwala lepiej je przetestować lub poszukać wsparcia społeczności, jeśli będziemy poszukiwali rozwiązania napotkanego problemu.

Czasem systemy operacyjne tworzone są przez producenta mikrokontrolera i udostępniane za darmo celem stosowania z jego produktami. Użycie takiego systemu powoduje jednak, że zostaniemy przywiązani na stałe do jednej firmy, co może być ryzykownym posunięciem.

Należy zaznaczyć, że wszystkie opisane minisystemy spełniają kryteria czasu rzeczywistego, co nie jest regułą w wypadku dużych systemów. Istotną kwestią jest również wsparcie dla języka C++, którym może pochwalić się niewiele z nich. W tabeli 1 przestawiono listę najbardziej popularnych systemów.

Większość z umieszczonych w tab. 1 systemów możemy uruchomić wykorzystując jedynie wewnętrzne zasoby pamięciowe mikrokontrolera jednoukładowego. Jedynymi wyjątkami są tutaj rozbudowane systemy, takie jak ECOS oraz uCLinux, które wymagają przynajmniej kilku MB pamięci zewnętrznej. Oprócz samego mikrojądra zazwyczaj udostępniane są dodatkowe zestawy bibliotek, takich jak: warstwa abstrakcji sprzętu HAL, stosy TCP/IP czy USB, obsługa systemów plików np. FAT itp. Generalnie większość z tych systemów jest bardzo do siebie podobna, więc dokonując wyboru będziemy musieli kierować się przede wszystkim dostępnością systemu dla danej platformy oraz obecnością dodatkowych bibliotek dla danego mikrokontrolera, np. obsługi USB. Przy zastosowaniach komercyjnych istotne jest również licencjonowanie oraz ewentualnie dostępność odpłatnego wsparcia. Konkurencja na tym polu jest jednak tak duża, że większość z nich jest dostępna nieodpłatnie wraz z kodami źródłowymi, natomiast licencje komercyjne, wymagające uiszczania dodatkowych opłat, stanowią nieliczne wyjątki. Należy również zastanowić się nad tym czy zawsze będziemy potrzebowali wszystkich dodatkowych bibliotek dostarczanych z systemem. Często lepszym wyborem będzie wykorzystanie jedynie jądra systemu oraz użycie oddzielnych bibliotek dostarczanych przez producenta mikrokontrolera. Podobnie jest ze stosem TCPIP, gdzie np. popularny stos LWIP daje się bardzo łatwo przenieść na dowolny system operacyjny i platformę niewielkim nakładem pracy. Dzięki modułowemu podejściu uzyskujemy dużo większą swobodę, jeśli chodzi o dobór oprogramowania.

Skupimy się teraz na bardzo krótkim omówieniu charakterystyki najbardziej popularnych systemów oraz ich charakterystyce.

ChibiOS/RT (http://www.chibios.org)

System operacyjny ChibiOS/RT jest stosunkowo nowym, dostępnym na licencji GPL. Projekt początkowo powstał jako hobbystyczny, jednak stopniowo zaczął się rozrastać się i obecnie jest rozwijany przez społeczność. Ma przy tym możliwość wykupienia wsparcia komercyjnego. Cechą charakterystyczną istotnie wyróżniającą go spośród innych systemów jest jego statyczna architektura, umożliwiająca tworzenie wszystkich obiektów systemu statycznie w trakcie kompilacji, bez konieczności wykorzystania funkcji alokacji pamięci. Również istotną cecha jest elastyczna konfiguracja, co umożliwia uzyskanie minimalnej funkcjonalności jądra już przy zajętości pamięci Flash rzędu 1,2 kB na platformie Cortex-M3. Jądro systemu zapewnia wielozadaniowość z wywłaszczeniem 128 priorytetów wątków, karuzelowe przełączanie zadań o takich samych priorytetach. Z mechanizmów komunikacji międzyprocesowej dostępne są: mutexy, zmienne warunkowe, semafory, zdarzenia, kolejki. Poza samym jądrem systemu, w późniejszym czasie dodano warstwę abstrakcji sprzętu HAL umożliwiającą tworzenie aplikacji niezależnie od platformy. Jako komponenty opcjonalne do wyboru są dostępne dwa stosy TCP/IP: popularny uIP oraz bardziej rozbudowanty LWIP. Dostępna jest również obsługa systemu plików FAT z wykorzystaniem biblioteki FATFS. System zawiera zestaw najbardziej popularnych i potrzebnych bibliotek przydanych podczas tworzenia własnych aplikacji.

FreeRTOS (http://www.freertos.org/)

System operacyjny FreeRTOS jest jednym z najbardziej popularnych systemów dostępnych dla mikrokontrolerów jednoukładowych. Jednocześnie jest to jeden z najstarszych projektów *open source* i pewnie jest znany większości czytelników. Projekt jest dostępny dla 31 typów mikrokontrolerów, zawiera wiele gotowych przykładów oraz obszerną dokumentację. Obsługiwanych jest również wiele rodzajów kompilatorów (w tym GCC). Istnieje możliwość wykupienia wsparcia technicznego, a ponadto możliwość nabycia komercyjnej odmiany systemu o nazwie SafeRTOS przeznaczonej dla aplikacji krytycznych pod względem bezpieczeństwa. Wersja komercyjna ma certyfikaty zgodności z normami IEC61508, EN32304, FDA510k. Projekt został w całości napisany w języku C, aby oprogramowanie zajmowało niewiele miejsca oraz było przenośne. Jądro systemu zapewnia wielozadaniowość z wywłaszczeniem oraz podstawowe mechanizmy komunikacji międzyprocesowej, takiej jak: mutexy, kolejki oraz semafory. Dodatkową cechą wyróżniającą system jest możliwość pracy w trybie *tick-less*, czyli bez przerwania zegarowego systemu. Ten tryb jest przeznaczony do obsługi urządzeń krytycznych ze względu na pobór mocy. Istotną cechą jest również możliwość pracy systemu na komputerze PC bezpośrednio na platformie Win32 lub POSIX, co umożliwia tworzenie oprogramowania bezpośrednio na komputerze PC doskonale ułatwiając w ten sposób pracę.

ISIX (http://bryndza.boff.pl/index.php?dz=rozne&id=isixrtos)

ISIX jest system mojego autorstwa. Z założenia został zoptymalizowany dla platform 32-bitowych, kompilatora GCC, pełnego wsparcia dla języka C++ oraz dla uzyskania minimalnej zajętości zasobów. Obecnie system może działać na architekturze Cortex-M3 oraz Cortex-M4 , również w wersji z jednostką zmiennoprzecinkową. Jeśli istnieje potrzeba użycia systemu na innej architekturze, to w łatwy sposób może być on dostosowany dla innych platform. Jądro systemu zapewnia wielozadaniowość z wywłaszczaniem, możliwość obsługi wielu priorytetów oraz szeregowanie zadań o takim samym priorytecie z użyciem algorytmu round-robin. Liczbę elementów komunikacji międzyprocesowej ograniczono do niezbędnego minimum: do dyspozycji mamy semafory, oraz kolejki.

Biblioteki zawierają również pełne wsparcie dla języka C++, włącznie z obsługą wyjątków oraz same funkcje systemu dostarczające interfejs C++ wraz z obsługą przestrzeni nazw. Dzięki użyciu tylko niezbędnych elementów system zajmuje jedynie kilka kB pamięci Flash i jest łatwy w użyciu. Dodatkowo, oprócz samego systemu, jest dostarczana biblioteka *libfoundation*

zawierająca generyczne algorytmy, niezależne od platformy oraz biblioteka libstm32 będąca odpowiednikiem *STM32 standard perhipheral library*. Formalnie wspomniane biblioteki są zupełnie niezależne od systemu i mogą być używane oddzielnie. Dodatkowo, dla systemu dostępny jest stos TCP/IP LWIP, a dla mikrokontrolerów STM32 STOS USB od ST wpierający tryby *Host* i *Device*. Zaimplementowano również bibliotekę obsługi systemu plików FAT32 bazująca na FATFS oraz warstwę abstrakcji dla kart SD wspierająca karty SDHC. System jest dostępny na licencji GPL z możliwością linkowania z własnymi aplikacjami bez konieczności publikowania kodu źródłowego.

MQX (http://www.freescale.com/webapp/sps/site/homepage.jsp?code=MQX_HOME&tid=vanMQX)

System operacyjny MQX jest przeznaczony dla mikrokontrolerów produkowanych przez firmę Freescale i dostępny jest na licencji komercyjnej. Podobnie jak inne systemy oparte o architekturę mikrojądra, zapewnia algorytm szeregowania oparty o wielozadaniowość z wywłaszczeniem oraz obsługą priorytetów. Zapewnia też podstawowe mechanizmy komunikacji międzyprocesowej, takie jak: wątki, semafory, kolejki komunikatów. Istotną zaletą dla użytkowników produktów tej firmy jest pełna integracja ze wszystkimi bibliotekami obsługi sprzętu przez co przygotowanie systemu do użycia jest bardzo łatwe. Dostępne są również zaawansowane biblioteki, takie jak: stos TCP/IP, obsługa systemu plików FAT oraz obsługa USB w trybach *Host* jak i *Device*. Cechą charakterystyczną jest również możliwość minimalistycznej konfiguracji zapewniającej podstawową funkcjonalność systemu już przy zajętości pamięci Flash rzędu 6 kB.

Contiki (http://www.contiki-os.org)

Contiki jest specjalistycznym systemem przeznaczonym dla aplikacji sieciowych, którego celem jest możliwość uruchomienia obsługi sieci na 8-bitowych urządzeniach o niewielkich zasobach sprzętowych, z naciskiem na optymalizację poboru energii. Jego głównym przeznaczeniem, są systemy czujników sieciowych. Umożliwia on uruchomienie aplikacji sieciowych już przy zajętości pamięci Flash rzędu pojedynczych kB, przy poborze energii rzędu miliwatów.

System operacyjny Contiki ma odmienną budowę od pozostałych. Szeregowanie zadań realizowaną jest w trybie wielozadaniowości kooperatywnej, bez wywłaszczania i jest oparte o koncepcję nazwaną *protothread*. Koncepcja ta opiera się o specjalistyczny system makr dając użytkownikowi wrażenie, że ma do czynienia z odrębnym procesem, a tak naprawdę jest to ukryta maszyna stanów, oparta o system zdarzeń. Wątek – zadanie jest pozbawiony stosu lokalnego zmuszając programistę do używania zmiennych globalnych. Należy tutaj podkreślić, że ten system nie jest systemem czasu rzeczywistego. Jego cechą charakterystyczną jest wsparcie zarówno dla protokołu IPv4, jak i jego następcy IPv6.

ECOS (http://ecos.sourceware.org/)

System operacyjny ECOS jest przeznaczony dla nieco bardziej rozbudowanych urządzeń wyposażonych w przynajmniej kilkaset kilobajtów pamięci operacyjnej, na których nie jest możliwe uruchomienie pełnych systemów na przykład Linuksa. Jądro systemu zapewnia mechanizm przełączania zadań z wywłaszczaniem oraz zaawansowane mechanizmy komunikacji międzyprocesowej. Dodatkowo, istotną cechą jest warstwa kompatybilności ze standardem POSIX, co umożliwia łatwe przenoszenie aplikacji przeznaczonych dla systemów: Unix, Linux, BSD. System dodatkowo ma zintegrowany stos TCP/IP pochodzący z FreeBSD. Niestety, ten stos nie jest na bieżąco aktualizowany i synchronizowany z kodem stosu FreeBSD, a ostatnia synchronizacja miała miejsce w 2001 roku, przez co system ma stare luki bezpieczeństwa oraz „exploity", za co jest krytykowany. Obecnie stracił mocno na znaczeniu z uwagi na to, że mikroprocesory z MMU oraz pamięci zewnętrzne bardzo potaniały. Ponadto układ z zewnętrzną pamięcią oraz zewnętrznymi magistralami traci wszystkie zalety projektów opartych o mikrokontrolery jednoukładowe. Rozwój systemu znacząco zwolnił i ostatnia wersja stabilna 3.0 pochodzi z roku 2009.

BeRTOS (http://www.bertos.org/)

BeRTOS jest kolejnym systemem modularnym, opartym o architekturę mikrojądra z obsługą wielozadaniowości z wywłaszczaniem oraz podstawowymi mechanizmami IPC, takimi jak: semafory, sygnały, kolejki komunikatów. Został on napisany w języku C i przeniesiony na najbardziej popularne układy rodziny ARM7TDMI, Cortex-M oraz AVR. Jego cechą charakterystyczną jest budowa modularna oraz rozbudowana warstwa abstrakcji sprzętu HAL zawierająca ogromną bazę sterowników urządzeń niespotykanych w innych systemach. Dodatkowo, oprócz warstwy abstrakcji sprzętu do dyspozycji mamy bibliotekę obsługi systemu plików FAT, obsługę autorskiego systemu plików BattFS, który jest przeznaczony dla małych nośników, takich jak pamięci EEPROM lub DataFlash. Jest wyposażony również w stos TCP/IP oraz obsługę dodatkowych protokołów, niespotykaną w innych systemach. Są to na przykład PocketBus czy AX25. Z systemem jest dostarczana także biblioteka umożliwiająca tworzenie prostych interfejsów graficznych. System jest dostępny na licencji GPL z możliwością linkowania własnych aplikacji bez konieczności publikowania ich kodu źródłowego.

Erika Enterprise (http://erika.tuxfamily.org/drupal/)

Erika Enterprise jest kolejnym systemem *open source*, który w 2012 roku otrzymał certyfikat OSEK/VDK. Podobnie jak i w innych systemach, mikrojądro zapewnia wielozadaniowość z wywłaszczaniem obsługę podstawowych mechanizmów IPC. Ma przy tym budowę modułową. W wersji minimalnej funkcjonalny kernel „zadowala się" pamięcią Flash rzędu 2 kB. Jego wyróżniającą cechą jest wsparcie dla architektury wielordzeniowej, które zazwyczaj nie jest spotykane w tej klasie systemów. Dla systemu, oprócz wsparcia społeczności, jest dostępne odpłatne wsparcie komercyjne.

FunkOS (http://funkos.sourceforge.net/)

System operacyjny FunkOS jest dostępny dla popularnych platform mikrokontrolerowych, takich jak ARM, AVR, MSP430. Jądro systemu zapewnia obsługę wielozadaniowości z wywłaszczaniem, z możliwością obsługi 255 priorytetów zadań oraz podstawowymi mechanizmami komunikacji IPC. Ponadto, jest dostępna warstwa abstrakcji sprzętu HAL, obsługa systemu plików FAT16/32 oraz biblioteka do budowy interfejsów graficznych. Jego interesującą cechą jest to, że dla języka C++ istnieje zupełnie odrębne jądro napisane w C++. System ma pełne wsparcie dla C++. Dostępny jest na licencji *open source – Sleepycat*.

Nucleus RTOS (http://www.mentor.com/embedded-software/nucleus/)

NucleusRTOS jest systemem o zamkniętej licencji, utworzonym przez oddział systemów wbudowanych firmy *Mentor Graphics*. Producent chwali się, że jego system pracuje w ponad 3 milionach urządzeń. Jest w pełni skalowany, a jego podstawową funkcjonalność możemy uzyskać już przy zajętości 13 kB pamięci Flash. System zawiera szereg bibliotek ułatwiających pracę programistom, takich jak: obsługa interfejsów I²C, SPI, USB2.0/USB3.0. Zawiera również zintegrowany stos TCP/IP. Ciekawą funkcjonalnością jest dostarczane przez producenta środowisko IDE oparte o Eclipse.

MicroC/OS-II (http://micrium.com/rtos/)

MicroC/OS-II jest systemem komercyjnym stworzonym przez firmę Micrium. Producent chwali się, że jego system jest wykorzystywany przez NASA do obsługi pojazdu Curiosity. Oparty na architekturze mikrojądra zapewnia wielozadaniowość z wywłaszczaniem, obsługę dowolnej liczby wątków (limitowaną wielkością dostępnej pamięci) oraz wsparcie dla mikrokontrolerów posiadających jednostkę ochrony pamięci MPU (nie mylić z MMU). System ma biblioteki TCP/IP, obsługę systemu plików FAT oraz SAFE oraz stos USB dla trybów *Host* i *Device*. Dostępne są również biblioteki dla obsługi CAN oraz Modbus. Tworzenie aplikacji ułatwia zintegrowane środowisko IDE oparte o Eclipse. System w większości został napisany w języku C, a w pełni funkcjonalny kernel zajmuje około 20 kB pamięci Flash.

uCLinux (http://www.uclinux.org/)

uCLinux jest osobnym forkiem jądra systemu Linux dla mikrokontrolerów niemających jednostki zarządzania pamięcią MMU. Jest to w pełni funkcjonalny kernel z dodaną emulacją jednostki zarządzania pamięcią MMU. Niemniej jednak, z uwagi na brak pamięci wirtualnej, wszystkie aplikacje uruchamiane są w ramach wspólnej przestrzeni adresowej razem z kernelem, przez co nie jest zapewniona pełna ochrona pamięci, jak w wypadku tradycyjnego jądra. uCLinux nie jest kompletnym systemem, a jedynie samym jądrem. Jako osobne projekty rozwijane są biblioteki towarzyszące np. ucLibc oraz dystrybucja formalnie stanowiąca system, która nosi nazwę ucLinux-dist. System do uruchomienia wymaga dostępnych kilku MB pamięci RAM oraz Flash na system plików, a więc nie jest możliwe jego uruchomienie na mikrokontrolerach jednoukładowych bez dołączenia zewnętrznych pamięci. Obecnie stracił mocno na znaczeniu z uwagi na powszechną dostępność tanich mikroprocesorów wyposażonych w jednostkę MMU, w związku z czym uruchamianie jądra Linuksa na mikrokontrolerach bez MMU straciło rację bytu.

Lucjan Bryndza, EP
l@boff.pl