Systemy operacyjne na urządzenia mobilne

Piotr Jastrzębski Piotr Laskowski Maciej Szarliński Tomasz Turski
studenci III roku infromatyki
Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego
Systemy Operacyjne 2008/09

22 stycznia 2009

Spis treści

1 Wstęp

Aby móc rozważać, czym charakteryzują się systemy na urządzenia mobilne, jakie przed nimi stoją wyzwania i ograniczenia spójrzmy na początek na skrótowy przegląd rozwoju urządzeń mobilnych: ich możliwości i przeznaczenia. Początki takich urządzeń sięgają lat 80tych XX wieku, kiedy to firma Psion prezentuje swoje pierwsze urządzenia SIBO, kontrolowane przez system operacyjny EPOC16. Urządzenia te przypominają zaawansowany kalkulator i posiadają funkcjonalność elektronicznego organizera. W związku z tym jak i również możliwościami obliczeniowymi tej maszyny (0,614 MHz, 16 KB RAM) system EPOC16 jest systemem jednozadaniowym, realizującym podstawowe funkcjonalności. Następnie w latach 90tych ubiegłego wieku następuje rozwój badań nad mobilnymi urządzeniami, jedną z firm zajmującą się takimi badaniami jest firma Apple, która w 1993 roku wypuszcza na rynku swoje urządznie Apple Newton. Parametry techniczne 20MHz, 640KB of RAM pozwalają na zastosowanie specjalnie przygotowanego na ten cel systemu operacyjnego NewtonOs, implementującego wiele usług znanych z późniejszych urządzeń. Dynamiczny rozwój rynku PDA (personal digital assistant - komputer kieszonkowy) następuje dopiero po 2000 roku kiedy to na rynku zaczyna przodować Palm z bogatą ofertą urządzeń Palm zarządzanych przez system operacyjny PalmOs. Przykładowy, zaprezentowany w 2001 roku, model Palm m505 oferuje sprzęt o parametrach: 33 MHz, 8 MB RAM i działa pod systemem PalmOs w wersji 4.0. Interesującym jest, iż system PalmOs do wersji 5 był systemem jednozadaniowym, nastawionym na interakcję z użytkownikiem co o dziwo na ówczesne potrzeby było satysfakcjonujące. Ostatecznie rynek urządzeń mobilnych skłania się ku rozwiązaniu określanemu jako "smartphone", czyli połączeniu funkcjonalności PDA i telefonu komórkowego w jednym urządzeniu. Najbardziej znanymi na tym polu producentami stają się firma RIM ze swoim urządzeniem BlackBerry (BlackBerry 8350, 2005 rok - 312 MHz, 16 MB RAM, BlackBerry OS), które jako urządzenie dla biznesu zachowuje charakter PDA, inne firmy natomiast stawiają na klienta masowego, proponując mu zaawansowaną multimedialność w ramach jednego urządzenia. Przykładowymi takimi urządzeniami są iPhone (2007 rok - 412 Mhz, 128 RAM, iPhone OS) i T-Mobile G1 (2008 rok - 528 Mhz, 192 RAM, Android).

1.1 Wprowadzenie

Urządzenia przenośne bezsprzecznie weszły w rzeczywistość dnia codziennego. Na dzień dzisiejszy wszyscy przyzwyczajeni jesteśmy do komunikacji bezprzewodowej, która to stała się głównym motorem napędowym rozwoju zaawansowanych urządzeń mobilnych. Dzięki rozwojowi komunikacji bezprzewodowej stanęła przed nimi zupełnie nowa rola: zamiast funkcji podróżnego organizera miały stać się pełnoprawnymi rezydentami sieci komputerowych, zarówno korporacyjnych jak i globalnej sieci Internetu. Ta drastyczna zmiana wymagała spojrzenia od nowa na podejście do projektowania systemów operacyjnych przygotowywanych na urządzenia mobilne, gdyż użytkownicy chcąc zachować dotychczasową wygodę użytkowania (interaktywność, responsywność, czas działania) jednocześnie oczekują funkcjonalności komputera osobistego (co wiąże się m.in. z wielozadaniowością, obsługą

wszelakiej maści multimediów).

1.2 Cena mobilności

Oczywistymi ograniczenia wiążą się z fizyczną charakterystyką jak waga i rozmiar, co przekłada się na ograniczone możliwości obliczeniowe i komunikacyjne. Kolejną rzeczą jest ograniczona ilość energii zasilającej, będącej jednym z najistotniejszych zasobów, co wymusza na projektantach jak najlepsze jej zagospodarowywanie.

1.3 Procesory

Możliwe ograniczenia, spotykane w procesorach stosowanych w urządzeniach mobilnych to:

- ograniczona programowalność - mobilne procesory posiadaj ograniczony zestaw instrukcji (w porównaniu z procesorami stacjonarnymi) ze względu na ograniczenia związane z poborem energii
- brak sprzętowego MMU, obsługi przerwań - wymusza to na projektantach systemu zastosowanie alternatywnych rozwiązań realizowanych programowo

Oczekiwania wobec procesorów stosowanych w urządzeniach mobilnych to:

- niski pobór energii ze względu na ograniczoną jej ilość
- relatywnie duża moc obliczeniowa
- dobra przepustowość operacji wejścia/wyjścia - np. zoptymalizowane instrukcje związane z wykorzystaniem sieci
- optymalizacja przetwarzania strumieni danych - ze względu na wymaganie sprawnej obsługi multimediów często procesory mobilne posiadaj specjalne instrukcje dla obsługi tej funkcjonalności

1.4 Procesory - najpopularniejsze architektury

Najczęściej spotykaną architekturą jest architektura RISC. Najpopularniejszymi procesorami/architekturami spotykanymi w dzisiejszych urządzeniach są:

- ARM (Advanced RISC Machine) – obecnie jest to najpopularniejsza architektura procesora. Na jego bazie powstał szeroko wykorzystywany XScale firmy Intel. Położono w nim nacisk na zwiększenie efektywności potokowości.
- MIPS – (Microprocessor without Interlocked Piped Stages) jest to architektura komputerowa (a w szczególności procesor typu RISC) rozwijana przez firm MIPS Technologies. Istnieje zarówno w wersji 32-, jak i 64-bitowej. Procesory MIPS stanowi jednostką centralną komputerów firmy SGI. Ponadto są szeroko stosowane w systemach

wbudowanych, głównie w urządzeniach opartych na systemie operacyjnym Windows CE. Szacuje się, że procesory MIPS stanowią jedną trzecią produkcji mikroprocesorów typu RISC.

- Crusoe i Efficeon – procesory opracowane przez firmę Transmeta. Wykorzystujące nowatorski system zarządzania energią (LongRun), umożliwiający dynamiczną i częstą zmianę napięcia i częstotliwości taktowania procesora w zależności od stopnia zapotrzebowania na moc obliczeniową zgłaszaną przez system operacyjny i uruchomione aplikacje. Crusoe i Efficeon są kompatybilne z procesorami typu x86 dzięki wykorzystaniu kodu pośredniego umożliwiającego tłumaczenie instrukcji procesorów x86 na natywne instrukcje VLIW.

1.5 Pamięć

Ze względu na zewnętrzne ograniczenia fizyczne (rozmiar urządzenia, ograniczone zasilanie, trudne warunki - wstrząsy) nie stosuje się pamięci masowych o wrażliwych mechanicznych elementach (takich jak np. dyski optyczne). W związku z tym powszechnym jest stosowanie pamięci SRAM jako pamięci operacyjnej dla aplikacji, pamięci flash do przechowywania kodu systemu operacyjnego i aplikacji. Jako pamięć masową stosuje się karty pamięci np. typu SD, co również jest istotne z punktu widzenia projektanta systemu operacyjnego (np. krótszy niż dla dysków optycznych czas dostępu do pamięci typu flash, system plików dostosowany do charakterystyki tego typu pamięci (kosztowna operacja clean)). Ponadto ze względu na (czasem zupełny brak pamięci innej niż operacyjna) stosunkowo mały rozmiar kodu systemu operacyjnego i aplikacji w systemach mobilnych przeważnie rezygnuje się z mechanizmu pamięci wirtualnej, jednocześnie (o ile wpierane przez sprzęt) zachowując inne mechanizmy jak stronicowanie, adresy logiczne czyli w praktyce rezygnując z możliwości wymiatania stron na dysk. Spotykane są również rozwiązania gdy ta sama pamięć jest wykorzystywana jako miejsce do składowania danych i uruchamiania programów, co również wymaga specjalnego podejścia. Podobnie wykorzystywane jest rozwiązanie uruchamiania programów "w miejscu" (czyli bez kopiowania kodu programu do pamięci operacyjnej w przypadku gdy treść programu na stałe rezyduje w pamięci charakteryzującej się krótkim czasem dostępu).

1.6 Funkcjonalność telefonu

Z racji łączenia funkcjonalności PDA i telefonu dobry mobilny system operacyjny powinien efektywnie i przeźroczyście obsługiwać wszystkie funkcje telefonu. Z związku z tym w mobilnych systemach operacyjnych implementuje się system software'owego real-time'u aby umożliwić komfortowe korzystanie z tychże funkcji. Wymaga to narzucenia wymagań na sprzęt odnośnie odpowiedniej responsywności (przewidywalny i ograniczony czas odpowiedzi).

2 Windows Mobile

2.1 Windows dla urządzeń mobilnych

Różnica między Windows CE a Windows Mobile:

- Windows CE - rdzeń systemu czasu rzeczywistego, bogaty zbiór ponad 700 komponentów,
- Windows Mobile - kompletna platforma przeznaczona na smartphone'y i pocketPC, zawiera własną powłokę i aplikacje.

Windows Mobile 5.0 jest oparty na komponentach z Windows CE 5.0 (są tego konsekwencje omówione dalej). Na starcie zawiera rozbudowaną funkcjonalność, np. obsługę poczty elektronicznej, pakiet biurowy, obsługę touchscreena, kamery.

2.2 Architektura jądra

Kod jądra systemu Windows CE 5.0 składa się z ponad 2,5 miliona linii kodu. Następna wersja ma ich o 56% więcej. Jest to jądro monolityczne, w odróżnieniu do mikrojądra dostarcza bogate środowisko programistyczne. Jądro stanowi warstwę pośredniczącą między aplikacjami użytkownika a sprzętem. Zwróćmy uwagę na:

- NK.exe – proces jądra,
- Coredll.dll – kody funkcji bibliotecznych wywoływanych przy wywoływania funkcji systemowych (np. `open()`), zostanie ona szerzej opisana w dalszej części.

2.3 Procesy i wątki. Szeregowanie

Windows Mobile 6.0 jest systemem wieloprocesowym i wielowejściowym, obsługuje jednocześnie do 32 procesów (takie ograniczenie nakłada Windows CE 5.0). Każdy proces posiada jeden lub więcej wątków.

2.3.1 Wątki

Wątek jest podstawową jednostką wykonawczą i podlega schedulingowi. Każdy wątek ma swój priorytet od 0 do 255 (nadajemy go przez funkcję `SetThreadPriority`). Wątki jednego procesu nie muszą mieć tego samego priorytetu. Im mniejsza wartość, tym wyższy priorytet w kontekście szeregowania. Wątki nawzajem nie wiedzą o swoim istnieniu, współdzielą dane procesu (wątku macierzystego). Aby zapobiec fragmentacji pamięci, każdy wątek posiada swój własny stos, którego maksymalną wysokość możemy ustawić poprzez parametr `dwCreationFlags`. Aby utworzyć wątek, wołamy funkcję

```
HANDLE CreateThread(LPSECURITY_ATTRIBUTES lpThreadAttributes,
```

```
DWORD dwStackSize, LPTHREAD_START_ROUTINE lpStartAddress,
LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId );
```

Każdy wątek jest w jednym z 5 stanów – `RUNNING, SLEEPING, SUSPENDED, BLOCKED` lub `TERMINATED`. Gdy żaden wątek nie jest gotowy do wykonania, scheduler wybiera wątek *idle*.

Ciekawostka. Dodatkowo, wątek może w swoim imieniu powołać włókna (ang. Fibers), które używają jego stosu i pewnego podzbioru rejestrów. Wątek sam jest odpowiedzialny za ich synchronizację.

2.3.2 Systemy real-time (RTOS)

- *hard real-time* - nieprzekraczalne deadliny (sprzęt medyczny),
- *soft real-time* - minimalizacja ilości przekroczonych terminów i sumarycznego opóźnienia.

Jeśli rozważamy obie klasy systemów operacyjnych w terminologii dwuwymiarowej przestrzeni, gdzie na jednej osi jest optymalność, a na drugiej przewidywalność, wówczas systemy *hard real-time* maksymalizują obie wartości. Resztę przestrzeni zajmują systemy *soft real-time* (kompromisy).

Windows CE został zaprojektowany jako system *hard real-time*.

- wątki o dostatecznie wysokim priorytecie mają określone deadline'y,
- realizacja przerwań także jest z góry ograniczona czasowo (*interrupt latency*),
- przerwania zegarowe co 1ms,
- zapobieganie *priority inversion*,
- różnorodność mechanizmów synchronizacyjnych.

Niektóre komponenty systemu wpływają negatywnie na tzw. *real-time performance*.

- moduł *GWES* (*Graphics, Windowing, Event Subsystem*) – czas oczekiwania wątku na sekcję nie jest z góry ograniczony, ponieważ moduł jest chroniony sekcją krytyczną,
- odwołania do systemu plików (problem j.w.),
- obsługa błędu braku strony również nie jest ograniczona czasowo z góry.

System udostępnia 2 programy służące do pomiarów opóźnień w realizacji zadań:

- OSBench.exe,
- ILTimer.exe.

2.3.3 Scheduling

Wątki o tym samym priorytecie są zorganizowane w kolejkę round-robin. Domyślny kwant czasu wynosi 100 ms, chyba że OEM postanowił inaczej.
Wątek o mniejszym priorytecie wykonuje się, gdy wątki o większym priorytecie są zablokowane. Gdy taki wątek stanie się gotowy do wykonania, poprzedni zostaje zawieszony.
To prowadzi do zagłodzenia. Aby po części temu zapobiec stosuje się mechanizm dziedziczenia priorytetu – jeśli wątek używa zasobów, na które czeka wątek o wyższym priorytecie, wówczas tymczasowo priorytet jest podwyższany, aby nie został wywłaszczony.
Przykład. Załóżmy, że nie ma dziedziczenia priorytetu. Mamy wątki 1,2 i 3, wątek 1 ma najwyższy priorytet, a 3 najniższy. Początkowo 1 i 2 śpią. Wątek 3 zabiera mutex wchodząc do sekcji krytycznej. Budzi się 1, wątek 3 zostaje wywłaszczony, ale nadal blokuje sekcję. Wątek 1 musi poczekać. Teraz wątek 3 kontynuuje wykonanie, ale budzi się wątek 2. Ten również czeka na sekcję. Wreszcie wątek 3 może opuścić sekcję i oddać mutex. Jeśli wątek 3 odziedziczyłby priorytet po 1, zwolniłby zasób szybciej.

2.3.4 Synchronizacja

Programista ma do wyboru szereg mechanizmów, oto kilka z nich:

- klasa `CriticalSection` (wzajemne wykluczanie),
- semafory,
- muteksy,
- kolejki komunikatów.

Standardowym mechanizmem synchronizacji między procesami są tzw. pliki mapowane do pamięci. O tej technice powiemy więcej później.

2.4 Syscalls

Każde wywołanie systemowe powoduje wyjątek, o którym informowane jest jądro. Wyznacza ono proces (*syscall process*), któremu powierza wykonanie funkcji.
Proces wywołuje funkcję biblioteczną, której kod znajduje się w **Coredll.dll**.
Przekazywanie wątku. Proces-wykonawca jest wykonywany przez wątek, którego posiadaczem był proces wołający. Wątek ten działa na poprzednim stosie. Po wykonaniu pracy, wątek powraca do procesu macierzystego i wykonuje kod będący za wywołaniem funkcji systemowej.

2.5 Zarządzanie pamięcią i energią

2.5.1 Podział pamięci

Bazuje na Windows CE 5.0 – jest to system 32-bitowy ze stronicowaniem (rozmiar strony 4KB). 4GB pamięci jest dzielone przez system (dostęp uprzywilejowany – Kernel Mode,

wyższe 2GB) i aplikacje użytkownika (User Mode, pozostałe 2GB). Przestrzeń użytkownika jest z kolei podzielona na

- p-ń aplikacji (0 - 03FF FFFF), wykorzystywaną jako środownisko bieżącego procesu i dynamicznych bibliotek (ROM DLLs, wykonywane w miejscu),
- Large Files Area (4200 0000 - 7FFF FFFF), używana do alokacji większych obszarów pamięci i mapowania plików (patrz: Mapowanie plików).

Przestrzeń użytkownika (dolne 2GB) są podzielone logicznie na 64 sloty po 32 MB każdy. **Slot 0.** Sloty 2-64 są przechowują dane o wszystkich procesach użytkownika. Kiedy proces jest wybierany do wykonania, jego dane są kopiowane do slotu 0. Są tu przechowywane dane o alokacjach i wątkach, zorganizowane w kilka stosów, aby uniknąć fragmentacji pamięci. **Slot 1** jest przeznaczony dla bibliotek ładowanych z pamięci ROM, które są współdzielone przez wszystkie procesy (np. COREDLL.dll z funkcjami bibliotecznymi).

2.5.2 Wyrównywanie. Alokacja

Mamy wyrównywanie do 64-kilobajtowych bloków, należy więc rezerwować jak najwięcej za każdym razem. Alokacje są już wyrównywane do 4 KB (= rozmiar strony).
Dlatego stosuje się łączenie bibliotek .dll – 3 biblioteki po 20KB zajęłyby 3 x 64 = 192 KB, a po połączeniu mieszczą się w 64 KB.
Większe alokacje. Każdemu procesowi przysługuje 32 MB, do alokacji większych niż 2MB używana jest pamięć z Large Files Area. Te obszary pamięci są współdzielone przez wszystkie procesy (zaproszenia zawierające adres), co wymaga synchronizacji.

2.5.3 Mapowanie plików

Innym mechanizmem służącym do alokowania większej ilości pamięci stanowią pliki mapowane do pamięci (*memory-mapped files*). Są one przechowywane w Large Files Area i można na nich wykonywać standardowe operacje jak na zwykłych plikach. Ta technika alokacji jest rekomendowana, umożliwia współdzielenie części danych przez procesy.

2.5.4 Małe podsumowanie i kilka wskazówek

Kilka ważnych zasad dotyczących zarządzaniem pamięcią w aplikacjach:

1. do 32 MB dla każdego procesu użytkownika,
2. do 32 procesów w systemie (tyle jest slotów),
3. zawsze sprawdzaj, czy alokacja się powiodła (jest ciasno, błąd braku wolnej pamięci zdarza się relatywnie częściej niż w systemach na architekturach desktopowych),
4. staraj się przewidywać i robić od razu większe rezerwacje (wyrówywanie do 64 KB), a następnie alokuj (wyrównywanie do 4 KB stron),

5. łącz mniejsze DLL w większe,
6. alokacje powyżej 2 MB mają miejsce w Large Memory Area,
7. używaj mapowanych plików zamiast dużych alokacji,
8. dziel pamięć pomiędzy procesy.

2.5.5 RAM a ROM

ROM – przechowuje podstawowe oprogramowanie i dane o użytkownikach, nie potrzebuje energii,
RAM – tymczasowe dane na potrzeby kodu, Mobile 5.0 potrzebuje 32 MB, natomiast w urządzeniach stosuje się 64 MB pamięci RAM.

Urządzenia o większej ilości RAM zużywają proporcjonalnie więcej energii. W przypadku ROM zużycie nie zależy od wielkości. Pamięć ROM (lub Flash ROM) jest droższa od RAM. Najczęściej spotykana konfiguracja to 64 MB pamięci RAM (pozostaje 32 MB dla użytkownika).

2.5.6 Persistent Storage

Strategia używana w Smartphone'ach od 2002 roku, wykorzystanie pamięci Flash ROM przypomina twardy dysk w architekturach desktopowych. Zalety:

- bezpieczeństwo - wszystkie dane przechowujemy w pamięci ROM, dzięki temu są one niewrażliwe na zaniki energii,
- żywotność - zanim wprowadzone PS, obowiązywała reguła "72-hour rule" (przy krytycznym stanie baterii (w praktyce przy 50%) urządzenie hibernowało, aby utrzymać dane przez 3 dni), dzięki obciążeniu ROM, pobór energii zmalał.

Problemy:

- czas dostępu – ROM 10 razy wolniejszy od RAM (trzeba to uwzględnić w aplikacjach),
- ograniczona ilość zapisów! (ok. 1M) – zapisy do ROM są buforowane w pamięci RAM i zapisywanie pełnymi blokami.

Wsparcie systemowe – do buforowania zapisów użyteczna jest klasa `MemoryMappedFileStream`.

2.5.7 Minimalizacja zużycia baterii w aplikacjach

Unikajmy:

- aktywnego oczekiwania (ogólnie – niepotrzebnego obciążania procesora),
- utrzymywania aktywnych procesów w tle (aplikacje powinny być aktywne tylko, gdy widzi je użytkownik – funkcja `LOWORD (wParam)` służy do sprawdzenia stanu aplikacji: czy jest w tle czy na wierzchu).

2.6 Bibliografia

- http://blogs.msdn.com/mikehall/archive/2007/01/17/windows-mobile-and-windows-emb
  aspx
- http://www.ergonet.pl/microsoftexchangedodatkowe/microsoftexchangedostepmobilny/
  WindowsMobile6/tabid/537/Default.aspx
- http://msdn.microsoft.com/en-us/library/aa454885.aspx
- http://www.real-time.org/hardandsoftrealtime.htm
- http://msdn.microsoft.com/en-us/library/aa909237.aspx
- http://en.wikipedia.org/wiki/Persistent_storage

3 Symbian OS

3.1 Historia i charakterystyka

Korzenie Symbiana sięgają do roku 1988, kiedy to firma Psion stworzyła system SIBO. System ten został zaprojektowany dla małych urządzeń przenośnych. Przykład takiego urządzenia jest widoczny na slajdzie.

Nazwa SIBO to akronim od "16-bit organizer".

Głównymi zaletami SIBO były: dobre zarządzanie energią, posiadanie lekkich i efektywnych aplikacji oraz dobra współpraca z PC i innymi urządzeniami przenośnymi.

Programowanie aplikacji dla SIBO było oparte na C, projektowane obiektowo i używało silnika aplikacji. Silnik ten pozwalał ustandaryzować API oraz skorzystać z abstrakcji obiektowej. Symbian odziedziczył idee silnika aplikacji.

W połowie lat 90, firma Psion rozpoczeła pracę nad nowym systemem dla swoich urządzeń przenośnych. W założeniach, system ten miał być 32 bitowy, obiektowy, multimedialny i niezależny od architektury. Efektem tej pracy był, napisany w C++ i od początku projektowany obiektowo, system EPOC. EPOC rozwiną idee silnika aplikacji i dodał pomysł serwerów zarządzających dostępem do funkcji systemowych i urządzeń. Okazało się, że wiele urządzeń może działać z nowym systemem.

Aby wykorzystać nowe możliwości rynku telefonów komórkowych, firma Psion razem z liderami branży telefonów komórkowych takich jak: Nokia, Ericsson, Motorola, Panasonic, stworzyli organizację o nazwie Symbian. Organizacja ta przejeła system EPOC i rozpoczeła jego rozwój pod nową nazwą Symbian OS. Symbian został stworzony dla, tak zwanych

"smartphon'ów" i jest wyspecjalizowany dla takich urządzeń. Symbian jest zorientowany obiektowo (odziedziczył to po EPOC'u). Kiedy system z rodziny UNIX stworzy deskryptor pliku by użyć go do wywołania funkcji systemowej open, Symbian stworzy nowy obiekt RFile i wywoła jego metodę open().

3.2 Struktura systemu - mikrojądro

Symbian oparty jest na mikrojądrze. Taka decyzja architektoniczna spowodowana jest małą ilością zasobów systemowych jakie są dostępne na telefonach. Szczególnie duże znaczenie ma ilość pamięci operacyjnej. Nowo załadowany kernel Symbiana w wersji 8 zużywa około 200KB pamięci operacyjnej. Jest to znacznie mniej niż potrzebne jest przeciętnemu Linuxowi do załadowania swojego jądra. Jest to od kilkunastu do kilkudziesięciu MB.

Kolejnym atutem tego rozwiązania jest możliwość zmiany obsługi jakiegoś zasobu bez rekompilacji jądra. Wystarczy wprowadzić odpowiednie zmiany do serwera odpowiadającego za ten zasób i przekompilowanie go. Sprawia to że systemy oparte na mikrojądrze są bardziej elastyczne ponieważ lepiej wspierają podłączanie nowego sprzętu podczas działania systemu.

Z drugiej strony jądra monolityczne oferują szybszy dostęp do zasobów. Spowodowane jest to narzutem związanym z ładowaniem kodu odpowiadającego za obsługę zasobu, który ma miejsce w przypadku mikrojądra.

Przykładem może być scenariusz w którym chcemy skorzystać z portu podczerwieni. System oparty na mikrojądrze musi najpierw załadować do pamięci wymagany kod i dopiero może zacząć wykonywać zadanie. System oparty na jądrze monolitycznym może od razu przejść do wykonywania zadania. Trzeba też pamiętać, że użycie mikrojądro zwiększa liczbę wywołań systemowych a co za tym idzie przełączeń kontekstu. Ma to z pewnością wpływ na wydajność systemu. Było to szczególnie widoczne we wczesnych wydaniach Symbiana.

3.3 Struktura systemu - warstwy

Symbian podzielony jest na 4 warstwy. Wewnętrzne warstwy implementują podstawowe funkcje systemowe, które wykonują się szybko i są najbardziej uprzywilejowane. To znaczy, że mają dostęp do wszystkich komponentów systemu kiedy tego potrzebują.

Bardziej zewnętrzne warstwy implementują coraz bardziej złożone funkcje i są coraz mniej uprzywilejowane. Nanokernel udostępnia podstawowe funkcje systemowe takie jak: scheduling i synchronizację operacji, obsługę przerwań oraz obiekty służące do synchronizacji takie jak mutexy i semafory. Ta warstwa nie implementuje żadnych złożonych operacji jak na przykład przydzielanie pamięci. Warstwa Symbian OS kernel implementuje wszystkie funkcje systemowe potrzebne reszcie systemu. W szczególności serwerom. Każda operacja na tym poziomie jest uprzywilejowaną funkcją, która korzysta z szeregu funkcji udostęp-

nianych przez nanojądro, w celu implementacji bardziej złożonego zadania. Do zadań tej warstwy należą: obsługa wątków urzytkownika, scheduling procesów, przełączanie kontekstu, zarządzanie pamięcią, ładowanie dynamicznych bibliotek, złożone obiekty synchronizacyjne oraz komunikacja między procesami. Wszystki funkcje w tej warstwie mogą zostać przerwane np. przez przerwanie. Warstwa serwerów jest typowa dla architektur opartych na mikrojądrze. Operacje, które nie są uprzywilejowane lub mają bardzo złożoną implementację, zostają wyniesione do serwerów, które odpowiadają za konkretny zbiór funkcjonalności taki jak np. praca z soketami. Ta warstwa znajduje się już w trybie użytkownika i funkcje w niej implementowane tylko sporadycznie używają funkcji systemowych.

Aplikacje z warstwy użytkownika działają prawie całkowicie w trybie użytkownika i tylko czasamiej korzystają z funkcji systemowych za pomocą serwerów lub wywołań systemowych.

3.4 Struktura systemu - budowa

Smartphone'y mają specyficzne wymagania. Wymagają nie tylko bogatego środowiska podobnego do tych na desktopy ale również szeregu usług czasu rzeczywistego jak np. obsługa rozmowy telefonicznej. W celu spełnienia tych wymagań Symbian posiada dość skomplikowaną architekturę. Na dole schematu widzimy prostokąt reprezentujący sprzęt telefonu. Szereg komponentów jądra komunikuje się bezpośrednio z warstwą hardware'u. Sterowniki urządzeń są podzielone na dwie części: Physical Device Driver i Logical Device Driver. PDD korzysta bezpośrednio ze sprzętu a LDD wystawia interfejsy urządzeń do wyższych warstw. Jądro może komunikować się bezpośrednio ze sprzętem za pomocą Application-Specific Standart Product. Na koniec, części Symbiana, które odpowiadają za usługi czasu rzeczywistego, mogą wchodzić w interakcję ze sprzętem. Memory model przedstawia jak zorganizowane są urządzenia i jak system z nich korzysta. Symbian OS kernel jest oparty na nanojądrze ale jest zupełnie odrębny od części, która odpowiada za obsługę usług czasu rzeczywistego. Nanokernel implementuje podstawowe funkcje systemu. Real-time OS i Personality layer zostały zaprojektowane do obsługi usług czasu rzeczywistego związanych z komunikacyjnym wykorzystaniem telefonów. RTOS implementuje funkcje związane z telefonią GSM.

3.5 Procesy - Active object

Wiadomo, że wątki zostały stworzone ponieważ procesy potrzebują relatywnie dużo zasobów systemowych. Sprzęt dla którego dedykowany jest Symbian charakteryzuje się ograniczonymi zasobami. Z tego powodu stworzone zostały Active object'y, czyli odchudzone wątki. Bardzo często wątki poboczne odpalane są po to by czekały na jakieś zdarzenie i obsługiwały je gdy nastąpi. Pozwala to by cały program nie usypiał. Ideą Active objectów jest stworzenie lżejszych wątków, które rozwiążą ten problem. Każdy Active object skupia się na obsłudze jakiegoś zdarzenia, który powoduje wstrzymanie programu np. oczekiwanie na operację wejscia-wyjścia.

Każdy Active object współpracuje z Active Schedulerem. Wszystkie Active object'y w jedym procesie współpracują z jednym Active Schedulerem. Odbywa się to na zasadzie wzorca projektowego ServiceToWorker, czyli Active Scheduler wie na jakie zdarzenie czeka jaki Active Object i gdy to zdarzenie ma miejsce to wywołuje odpowiednią funkcję tego Active object'u. Skutkuje to tym, że wszystkie Active object'y w jednym procesie zachowują się jak jeden wątek ponieważ są obsługiwane przez ten sam Active Scheduler.

3.6 Procesy - Active object cz.2

Widzimy na slajdzie diagram przepływu sterowania.

1. Główny program tworzy Active Scheduler.
2. Główny program tworzy Active Objecty i dodaje je do Active Schedulera, który pamięta je w kolejce.
3. Główny program wywołuje IssueRequest ActiveObjectu.
4. Wywołana funkcja ustawia pole iActive w ActiveObjectcie na true i uruchamia service na odpowiedĽ którego będzie czekał ActiveObject. Zmianiany jest przy tym iStatus ActiveObjectu na KRequestPending.
5. Gdy zakończy się wywoływany service to zmienia on iStatus ActiveObjecta który na to czeka i informuje ActiveSchedulera.
6. ActiveScheduler przeszukuje wszystkie ActiveObjecty w kolejce i dla każdego który ma iActive = true i iStatus != KRequestPending ustawia mu iActive na false i wywołuje RunL tego objektu.

3.7 Procesy - Scheduling i synchronizacja

W Symbianie procesy mogą mieć jeden z 64 priorytetów. Aby przyspieszyć wybieranie procesu do uruchomienia, Symbian posiada 64 kolejki w których trzyma procesy gotowe do przydzielenia procesora o danym priorytecie. Szybkie znajdowanie niepustej kolejki zapewnia 64 bitowa maska.

Symbian jest systemen soft real-time. To znaczy, że manipulując priorytetami procesów, system ustawia na początku procesy czasu rzeczywistego w kolejności rosnących deadlinów a następnie zwykłe procesy. Taki scheduling nazywa się static, monotonic scheduling. Symbian posiada dwa poziomy synchronizacji. Na poziomie nanojądra dostępne są semafory i muteksy, które synchronizują podstawowe operacje. Na bazie objektów synchronizujących z nanojądra zbudowane są semafory i muteksy poziomu Symbian OS kernela. W Symbianie jeżeli dwa procesy o róznych priorytetach czekają na muteks to pierwszy dostanie go proces o wyższym priorytecie. Aby zapobiec deadlock'om, Symbian robi dwie rzeczy. Po pierwsze, muteksy nie są przyznawane procesom aż do ostatniego momentu, co daje

możliwość procesom o wyższym priorytecie wyprzedzić proces o niższym. Po drugie, jeżeli proces o niższym priorytecie trzyma muteks na który czeka proces o wyższym priorytecie, to priorytet procesu o niższym priorytecie zostaje podniesiony do priorytety procesu o wyższym priorytecie.

3.8 Zarządzanie pamięcią

Symbian jest 32 bitowym systemem więc może adresować do 4GB pamięci. Jak widać na załączonym obrazku, Symbian korzysta z dwupoziomowego tablicowania stron. Pierwsze 12 bitów adresu logicznego jest indeksem w TTBR czyli w translation table-base register. Następne 8 bitów to indeks w konkretnej tablicy stron. Ostatnie 12 bitów to indeks w stronie pamięci. Aplikacje są ładowane w całości do pamięci w momencie ich uruchomienia. System trzyma listę wolnych ramek pamięci i przydziela je do stron gdy jest taka potrzeba. Jeżeli nie ma wolnych ramek a istnieje potrzeba przydzielenia strony to podnoszony jest wyjątek. Symbian nie może nadpisać ramki używanej przez inną aplikację nawet jeżeli jest ona uśpiona. Jest to spowodowane brakiem swappingu co skutkuje brakiem miejsca do którego możnaby skopiować zawartość nadpisywanej ramki.

3.9 Model komunikacji

Model komunikacji składa się z trzech warst. Warstwa Sterowników Urządzeń dzieli się na dwie części. Pierwsza, LDD, wystawia standardowy interfejs niezależny od urządzenia, z którego korzysta wyższa warstwa. Druga, PDD, implementuje standardowy interfejs wystawiany przez LDD na konkretnych urządzeniach. Warstwa Protokołu podzielona jest na 4 części. Moduł CSY odpowiada za niskopoziomową komunikację z urządzeniem i łączy się bezpośrednio z PDD. Moduł TSY odpowiada za obsługę funkcji związanych z telefonią GSM. Moduły PRT implementują protokuły takie jak TCP/IP czy Bluetooth. MTM czyli Message Type Module, obsługuje wszelkiego typu wiadomości.

3.10 Bibliografia

- "Smartphone Operating System Concepts with Symbian OS" Michael J. Jipping
- "The Symbian OS Architecture Sourcebook" Ben Morris

4 Google Android

4.1 Co to Google Android?

- **Android**
  Platforma opensource dla telefonów komórkowych oparta na systemie Linux, ogłoszona przez Google i inne firmy zrzeszone w Open Handset Alliance. Pozwala programistom tworzyć aplikacje w Javie przy użyciu bibliotek opracowanych przez Google.

- **Open Handset Alliance**
  Konsorcjum założone 5 listopada 2007, w którego skład wchodzi 34 firm z branży telekomunikacyjnej, sprzętowej i programistycznej, między innymi: Google, Qualcomm, HTC, Intel, Samsung, Motorola, Sprint czy Texas Instruments. Jego celem jest rozwój otwartych standardów dla urządzeń mobilnych.

4.2 Historia Androida

- **lipiec 2005**
  Google zakupiło Android Inc., niewielką firmę z Kalifornii.

- **lipiec 2007**
  Doniesienia prasowe o tym, że Google pracuje nad mobilnym systemem operacyjnym.

- **sierpień 2007**
  The Wall Street Journal informuje, że Google chce wejść na rynek urządzeń mobilnych.

- **październik 2007**
  Network World donosi, iż telefon Google to otwarty system operacyjny dla telefonów komórkowych, a nie konkretne urządzenia, jak iPhone.

- **listopad 2007**
  5. listopada 2007 założono Open Handset Alliance, konsorcjum w którego skład wchodzą Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, Sprint Nextel oraz NVIDIA, z myślą o rozwoju otwartych standardów dla telefonii mobilnej. Wraz z ogłoszeniem powstania OHA zaprezentowano platformę Android.

- **luty 2008**
  W ramach Mobile World Congress 12 lutego 2008 Google zaprezentował trzy działające prototypy wyposażone w Androida.

4.3 Cechy Androida

- **Ekran**
  Platforma obsługuje różne rozdzielczości ekranów, w szczególności zarówno VGA jak i mniejsze. Biblioteki umożliwiają wykorzystanie zarówno grafiki 2D jak i 3D w oparciu o OpenGL ES 1.0.

- **Przechowywanie danych**
  Wykorzystanie Systemu Zarządzania Bazą Danych SQLite.
  SQLite - to system zarządzania bazą danych oraz biblioteka C implementująca taki system, obsługująca język SQL (ang. Structured Query Language). Biblioteka

implementuje silnik SQL, dając możliwość używania bazy danych bez konieczności uruchamiania osobnego procesu RDBMS, dlatego jest często wykorzystywana w systemach wbudowanych.

- **Komunikacja**
  Android wykorzystuje wiele technologii komunikacyjnych, m.in. GSM, CDMA, Bluetooth, EDGE, 3G oraz Wi-Fi.

- **Wiadomości**
  SMS, MMS, XMPP (wykorzystujące XML, w oparciu o protokół Jabbera).

- **Przeglądarka internetowa**
  Android używa wbudowanej przeglądarki opartej o WebKit.
  WebKit - silnik przeglądarki internetowej rozwijany na zasadach otwartego oprogramowania. i umożliwiający wyświetlanie stron internetowych. Był wykorzystywany pierwotnie w przeglądarce internetowej Safari firmy Apple.

- **Multimedia**
  Android obsługuje formaty takie jak MPEG-4, H.264, MP3 oraz AAC, JPEG, PNG, GIF.

- **Virtual machine**
  Oprogramowanie napisane w języku Java jest kompilowane do kodu Dalvika i uruchamiane przez Maszynę Wirtualną Dalvika, będącą wyspecjalizowaną maszyną wirtualną zaprojektowaną specjalnie dla urządzeń mobilnych. Maszyna ta nie jest kompatybilna z JVM.

- **Dodatkowe urządzenia**
  Android potrafi w pełni korzystać z aparatu fotograficznego oraz kamery, ekranu dotykowego, GPS, kompasu, sensorów motorycznych i akceleratorów grafiki 3D.

- **Środowisko programistyczne**
  Zawiera emulator, debugger, diagnostykę użycia pamięci, profiler wydajności oraz wtyczka dla Eclipse IDE

- **Architektura**
  Obecnie system Android pracuje na jądrze Linuxa dla architektury ARM, jednak w planach na 2009 rok jest wprowadzenie podstawowego wsparcia dla architektury x86.

4.4 Architektura systemu Android

- **Applications**
  System Android będzie dostarczany z zestawem podstawowych aplikacji, takich jak: klient e-mail, program do obsługi wiadomości SMS, kalendarz, mapa, przeglądarka internetowa, kontakty i inne. Wszystkie aplikacje pisane są w technologii Java.

- **Application Framework**
  Developerzy piszący aplikacje na platformę Android mają pełny dostęp do API używanego przez aplikacje dostarczane z systemem. Architektura aplikacji została zaprojektowana tak, by zwiększać stopień reużywalności komponentów. Każda aplikacja może dodawać swoje własne komponenty, jak i używać komponentów innych aplikacji. Mechanizm ten pozwala użytkownikom na zastępowanie dowolnych komponentów swoimi własnymi.

  Bazą wszystkich aplikacji jest zbiór usług i systemów zawierający:

  - Views - zestaw kontrolek (Views), które mogą zostać użyte przy tworzeniu aplikacji, takich jak: lisy, belki, pola tekstowe, przyciski a nawet przeglądarka internetowa,
  - Content Providers - system pozwalający na wymianę danych pomiędzy aplikacjami,
  - Resource Manager - system umożliwiający dostęp do zasobów,
  - Notification Manager - mechanizm pozwalający aplikacjom na wyświetlanie komunikatów w pasku statusu,
  - Activity Manager - menedżer aplikacji.

- **Libraries**
  Android zawiera zestaw bibliotek C/C++ używanych przez różne komponenty systemu. Ich funkcjonalności są dostępne dla programistów poprzez framework. Bazowy zestaw składa się m.in. z bibliotek:

  - System C library,
  - Media Libraries,
  - Surface Manager,
  - LibWebCore,
  - SGL,
  - biblioteki 3D,
  - FreeType,
  - SQLite.

- **Android Runtime**
  Android zawiera zestaw bazowych bibliotek, posiadających większość funkcjonalności dostępnych w podstawowych bibliotekach języka Java. Każda aplikacja jest uruchamiana jako osobny proces, wraz z jej własnym egzemplarzem wirtualnej maszyny Dalvik. Dalvik działa w oparciu o jądro Linuxa w zakresie niskopoziomowych funkcjonalności, takich jak zarządzanie wątkami czy niskopoziomowe zarządzanie pamięcią.

- **Linux Kernel**
  Android wykorzystuje jądro Linuksa w wersji 2.6 dla takich zadań, jak: bezpieczeństwo, zarządzanie pamięcią, zarządzanie procesami, obsługa sieci i sterowniki. Jądro jest warstwą przejściową pomiędzy sprzętem a oprogramowaniem.

4.5 Android Runtime

- **Dalvik**
  Wirtualna maszyna Javy, przeznaczona dla urządzeń o ograniczonych zasobach sprzętowych (pamięć i moc procesora).
- Każda aplikacja jest uruchamiana jako osobny proces, wraz z jej własnym egzemplarzem wirtualnej maszyny Dalvik.
- Dalvik kompiluje pliki Java do formatu .dex (Dalvik Executable).
- Dalvik działa w oparciu o jądro Linuxa w zakresie niskopoziomowych funkcjonalności, takich jak zarządzanie procesami, wątkami czy niskopoziomowe zarządzanie pamięcią.

4.6 Linux Kernel

- Linux Kernel 2.6.25 for ARM
  bezpieczeństwo, zarządzanie pamięcią, zarządzanie procesami, obsługa sieci i sterowniki. Wspiera architekturę ARM od wersji V5T.
- Do obsługi androida wykorzystuje się szereg rozszerzeń:
  alarm, ashmem, binder, power management, low memory killer, kernel degugger, and logger.
- Obsługa systemu plików FAT32.
- Obsługa TCP/IP (TCP, UDP, etc).
- Obsługiwany sprzęt
  Platforma będzie działać na większości środowisk Linuksowych, bazujących na architekturze ARM
  - minimum 128 MB RAM
  - minimum 256 MB pamięci Flash
  - 802.11 b/g Wi-Fi
  - Standardowy interfejs USB, wraz z USB 2.0
  - Bluetooth 2.0
  - Aparat do robienia zdjęć i nagrywania filmów
  - Gniazdo pamięci

4.7 Model aplikacji

- **Pakiet Android**
  Pakiet Android (.apk) jest plikiem zawierający kod i zasoby aplikacji. Pakiety, to pliki, w których rozpowszechniane i pobierane są aplikacje instalowane przez użytkowników.

- **Task**
  Task (zadanie) to część pakietu, postrzegana przez użytkowników jako "aplikacja", która może zostać uruchomiona: zadanie ma swoją ikonę w interfejsie użytkownika, przy użyciu, której można je uruchomić.

- **Proces**
  Proces jądra, w którym uruchomiona jest aplikacja. Zwykle cały kod z pakietu.apk jest uruchamiany jako jeden proces, możliwe jest jednak podzielenie go na kilka części składowych aplikacji: activity, receiver, service, provider.

  Główne zalety używania procesów:

  - podniesienie stabilności i bezpieczeństwa systemu poprzez umieszczanie niestabilnego lub niezabezpieczonego kodu w osobnych procesach,
  - redukowanie narzutów systemowych przez uruchamianie kodu kilku aplikacji w jednym procesie,
  - wsparcie systemu w zarządzaniu zasobami poprzez umieszczanie kodu o dużym zapotrzebowaniu na zasoby w osobnym procesie, który może zostać zatrzymany niezależnie od reszty aplikacji.

- **Wątek**
  W ramach każdego procesu działa jeden lub więcej wątków. W większości przypadków, Android unika tworzenia dodatkowych wątków w procesie, pozostawiając aplikację jednowątkową, jeżeli nie tworzy ona sama nowych wątków.

4.8 Architektura ARM

Architektura ARM (Advanced RISC Machine) jest 32-bitową architekturą procesorów typu RISC. Jest on szeroko stosowana w systemach wbudowanych i systemach o niskim poborze mocy, ze względu na ich energooszczędność.

Procesory ARM są używane między innymi w: dyskach twardych, telefonach komórkowych, routerach, kalkulatorach, a nawet w zabawkach dziecięcych.

Moc obliczeniowa procesorów ARM umożliwia instalacje na tym procesorze systemu operacyjnego, z zaimplementowanymi mechanizmami wielowątkowości, z możliwością wykorzystania zawartego w systemie stosu TCP/IP czy systemu plików (np. FAT32). Powstało

wiele takich systemów: Windows CE, NUTOS, i wiele dystrybucji Linuksa opatrzonym hasłem embedded (Embedded Debian, Embedded Ubuntu).

4.9 Bibliografia

- http://www.youtube.com/v/eNzkT9usvSY\&hl=en\&fs=1\&autoplay=1,
- http://pl.wikipedia.org/wiki/Android_(platforma),
- http://source.android.com/,
- http://code.google.com/intl/pl-PL/android/intro/appmodel.html,
- http://www.androidal.pl/,
- http://code.google.com/intl/pl-PL/android/,
- http://www.idg.pl/news/330608/Google.Android.aktualizacja.w.styczniu.2009.r.html,
- http://gphone.pl/,
- http://pl.wikipedia.org/wiki/Architektura_ARM,
- http://en.wikipedia.org/wiki/Dalvik_virtual_machine,
- http://source.android.com/release-features.

5 Podsumowanie

Podsumowaniem niech będzie porównanie przedstawionych systemów operacyjnych i rozwiązań które dostarczają w celu sprostania następującym kryteriom.

5.1 Niezawodność i interaktywność

- rodzaj jądra determinuje różne zalety i wady wynikające z tak wybranego rozwiązania
  - jądro monolityczne (tańsze podstawowe operacje, mniej ładowania modułów)
  - mikrojądro (serwery, izolacja kodu aplikacji, mniejszy rozmiar kodu systemu)
  - jądro hybrydowe
- izolacja aplikacji w procesie
  - umożliwia izolację ewentualnych błędów
  - daje możliwość wykrycia i zakończenia "zasobożernych" aplikacji

– software real-time stosowany do obsługi funkcjonalności telefonu (np. RTOS)

– standardowe mechanizmy (MMU, TLB, stronicowanie)

– wysokopoziomowe framework'i dla tworzenia aplikacji, umożliwiające ściślejszą kontrolę dostępem aplikacji do zasobów systemu

5.2 Bezpieczeństwo

Pierwszym i podstawowym z czynników jest rodzaj jądra (determinuje on jak w jakim stopniu oprogramowanie zewnętrzne może "namieszać" w jądrze systemu) i dostarczane przez nie możliwe tryby pracy np. tryb użytkownika, jądra, uprzywilejowany. Inną metodą na zapewnienie bezpieczeństwa stał się mechanizm podpisywania aplikacji, polegajacy na tym, iż dowolna aplikacja, którą chcemy uruchomić na urządzeniu musi wpierw zostać podpisana przez zewnętrzą firmę która wpierw weryfikuje bezpieczeństwo użycia tego programu. Rozwiązanie te jest stosowane w Androidzie, Symbianie i Iphone. Ponadto system Android dostarcza dodatkowe mechanizmy jak:

- mechanizm zezwoleń dla pakietu/aplikacji (aplikacja przy instalacji otrzymuje prawa do korzystania z wybranych zasobów i nie może korzystać z żadnych innych),
- mechanizm zezwoleń dostępu do URI (aplikacja może otrzymać jednorazowe pozwolenie na dostęp do zasobu zidentyfikowanego za pomocą URI),
- mechanizm nadawania każdemu pakietowi unikalnego UID, dzięki czemu domyślnie każdy pakiet jest właścicielem tworzonych przez niego danych i tylko on ma do nich dostęp.

5.3 Łatwość programowania

Na koniec rozpatrzmy omawiane systemy pod kątem łatwości programowania aplikacji dedykowanych dla danego systemu. Jedną z zalet ułatwiających tą czynność jest otwartość kodu źródłowego danego systemu. Tak jest w przypadku systemu Android, w planach jest opublikowanie kodu źródłowego Symbiana. Wszystkie systemy dostarczają SDK, IDE i emulatory wspomagające przygotowywanie aplikacji na dane systemy. Również często spotykanym (np. w Windows Mobile i Android) jest dostarczanie wysokopoziomowych framework'ów ułatwiających i przyśpieszających pracę. Ostatnim czynnikiem jest ilosć wspieranych języków programowania od jednego (java w Androidzie), dwóch (C++, C# w Windows Mobile) po kilka (C++, Java, Python, VisualBasic w Symbianie).

Istnieją również wady poszczególnych systemów, zdecydowanie utrudniające i zniechęcające do tworzenia aplikacji dedykowanych pod dany system. Jedną z takich wad jest wymieniony wcześniej wymóg podpisywania aplikacji (co jest przeważnie płatne) ograniczający możliwość rozwoju bezpłatnych aplikacji lub aplikacji rozwijanych przez małe firmy. Również dużą przeszkodą może być niespójność różnych wersji danego systemu, tak jak to dzieje

się w przypadku platform Symbiana przygotowywanych przez poszczególnych producentów telefonów. Na koniec warto wspomnieć o Darviku - wirtualnej maszynie Javy stosowanej w Androidzie, która mimo zapewnień nie zachowuje kompatybilności ze standardową wirtualną maszyną dostarczaną przez firmę Sun.

5.4 Bibliografia

Także do rozdziału **Wstęp**.

- praca magisterska "Aplikacje na urządzenia mobilne typu smartphone", Marcin Przekop, September 2007,
- `http://www.pcworld.pl/artykuly/48813_1/PDA.pecet.do.kieszeni.html`,
- `http://www.computer.org/portal/site/dsonline/menuitem.9ed3d9924aeb0dcd82ccc6716bb index.jsp?&pName=dso_level1&path=dsonline/topics/os&file=MobileOS-Nov03.xml&xsl=article.xsl&#articles`.