МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРАСНОДОНСЬКИЙ ПРОМИСЛОВО ЕКОНОМІЧНИЙ КОЛЕДЖ

Реферат з предмету: "Операційні системи"

На тему: "Короткі характеристики найбільш поширених ОСРЧ***"***

Студента групи 1ОКІСМ-06

Петренко Михайла

Перевірила: Дрокіна Т.М.

Краснодон 2009

Зміст

1. VxWorks

2. QNX Neutrino RTOS

3. RTEMS

4. ChorusOS

5. Розширення реального часу для Windows NT

5.1 RTX для Windows NT

5.2 INtime

2.5.1 Microsoft Windows Embedded

6. TinyOS

7. OSEK / VDX

8. OSE RTOS

9. Contiki

10. pSOS

11. INTEGRITY

12. LynxOS

13. Microware OS-9

14. GRACE-OS

15. C EXECUTIVE

16. CMX-RTX

16.1. CMX-TINY +

17. Inferno

## 1. VxWorks

Операційні системи реального часу сімейства VxWorks корпорації WindRiver Systems призначені для розробки програмного забезпечення (ПО) вбудованих комп'ютерів, що працюють в системах жорсткого реального часу [VxWorks]. Операційна система VxWorks має крос-засобами розробки програмного забезпечення (ПО), тобто розробка ведеться на інструментальному комп'ютері (host) у середовищі Tornado для подальшого її використання на цільовому комп'ютері (target) під керуванням системи VxWorks.

Операційна система VxWorks має архітектуру клієнт-сервер і побудована у відповідності з технологією мікроядра, тобто на самому нижньому безперервному рівні ядра (WIND Microkernel) обробляються тільки планування завдань та управління їх взаємодією / синхронізацією. Вся інша функціональність операційного ядра - управління пам'яттю, введенням / виведенням і пр. - забезпечується на більш високому рівні і реалізується через процеси. Це забезпечує швидкодію і детермінованість ядра, а також маштабованість системи.

VxWorks може бути скомпонована як для невеликих вбудованих систем з жорсткими обмеженнями для пам'яті, так і для складних систем з розвиненою функціональністю. Більше того, окремі модулі самі є маштабованими. Конкретні функції можна прибрати при збірці, а специфічні ядерні об'єкти синхронізації можна опустити, якщо додаток в них не потребує.

Хоча система VxWorks є конфігурується, тобто окремі модулі можна завантажувати статично або динамічно, не можна сказати, що в ній використовується підхід, заснований на компонентах. Всі модулі побудовані над базовим ядром і спроектовані таким чином, що не можуть використовуватися в інших середовищах.

Ядро VxWorks володіє наступними параметрами:

кількість завдань не обмежено,

число рівнів пріоритетів завдань - 256,планування завдань можливо двома способами - витіснення за пріоритетами і циклічне,

засобами взаємодії завдань служать черги повідомлень, семафори, події і канали (для взаємодії задач всередині CPU), сокети і віддалені виклики процедур (для мережевої взаємодії), сигнали (для керування винятковими ситуаціями) і колективна пам'ять (для розділення даних),

для управління критичними системними ресурсами забезпечується кілька типів семафорів: виконавчі, обчислювальні (counting) і взаємно виключають з пріоритетним спадкуванням,

підтримується детермінована перемикання контексту.

У VxWorks забезпечується як заснований на POSIX, так і власний механізми планування (wind scheduling). Обидва варіанти включають витісняється і циклічне планування. Різниця між ними полягає в тому, що wind scheduling застосовується на системному базисі, в той час як алгоритми POSIX-планування застосовуються на базисі процес-за-процесом.

У VxWorks всі завдання системи і додатків поділяють єдине адресний простір, що загрожує порушенням стабільності системи через несправність будь-якої програми. Необов'язковий компонент VxVMI дає можливість кожному процесу мати свою власну віртуальну пам'ять.

Щоб досягти швидкої обробки зовнішніх переривань, програми обробки переривань (ISRs - interrupt service routines) у VxWorks виконуються в спеціальному контексті поза контекстів потоків, що дозволяє виграти час, який зазвичай витрачається на перемикання контекстів. Слід зазначити, що C-функція, яку користувач приєднує до вектора переривання, насправді не є фактичною ISR. Переривання не можуть безпосередньо звертатися до C-функцій. Адреса ISR запам'ятовується в таблиці векторів переривань, яка викликається апаратно. ISR виконує якусь початкову обробку (збереження регістрів і підготовку стека), а потім викликається C-функція, яка була приєднана користувачем.

VSPWorks [VSPWorks] - це дуже популярна і досить потужна ОС на основі VxWorks. VSPWorks спроектована спеціально для систем, заснованих на DSP. Вона забезпечує багатозадачність з пріоритетами і підтримку швидких переривань на процесорах DSP і ASIC. ОСРВ VSPWorks слід моделі єдиного віртуального процесора, що значно спрощує розподіл програм багатопроцесорні системи, зберігаючи при цьому продуктивність жорсткого реального часу. VSPWorks є модульною і маштабованої.

ОСРВ VSPWorks має багатошарової структурою, що служить хорошою основою для абстрагування та переносимості. Центром системи служить сильно оптимізоване наноядро (nanokernel), яке здатне керувати сукупністю процесів. Нижче наноядра знаходяться програми, які обслуговують переривання, вище наноядра розташовується Мікроядро, яке управляє багатозадачному режимі з пріоритетами C / C + + завдань.

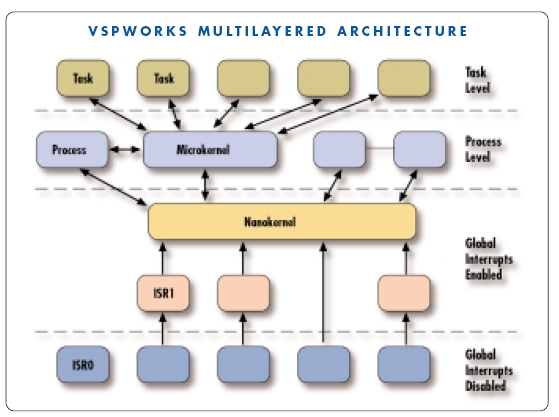


Рис.1. Багатошарова архітектура VSPWorks.

Управління переривань має два рівні. Нижній рівень (рівень 1) використовується для обробки апаратних переривань. Під час обробки таких переривань всі інші переривання блокуються. Код, що виконуються на цьому рівні, написаний на мові асемблера, і відповідальність за збереження відповідних регістрів в стеку лягає на програміста. На цьому рівні може бути оброблено переривання, яке вимагає малого часу для обробки. Якщо обробка переривання є більш складною й потребує більшого часу, то переривання обробляється на більш високому рівні (рівень 2), де дозволено переривання переривання і, таким чином, вони можуть бути вкладеними. Перехід на більш високий рівень переривань відбувається по системному викликом.

Процеси наноядра (рівень 3) пишуться на мові асемблера і мають скорочений контекст (тобто використовують менше регістрів). Ці процеси можуть бути завантажені і розвантажено з процесора дуже швидко. Кожному процесу присвоюється пріоритет. Рівень 3 ідеальний для написання драйверів для інтерфейсів апаратури низького рівня.

Мікроядро знаходиться на рівні 4. Мікроядро написано на мові C і має понад 100 сервісів. Обробка завдань на цьому рівні ведеться в режимі пріоритетного переривання, і планування управляється пріоритетами.

Мережеві засоби. VxWorks підтримує всі мережеві засоби, стандартні для UNIX: TCP / zero-copyTCP / UDP / ICMP / IP / ARP, SLIP / CSLIP / PPP, Sockets, telnet / rlogin / rpc / rsh, ftp / tftp / bootp, NFS (Network File System) (клієнт і сервер). У мережеві засоби для VxWorks входять також функції, необхідні при розробці пристроїв, що підключаються до Internet: IP multicasting рівня 0,1 або 2; long fat pipe; CIDR (Classless Inter-Domain Routing); DHCP (Dynamic Host Configuration Protocol) в конфігураціях server, client і relay agent; DNS client (Domain Naming System); SNTP (Simple Network Time Protocol). VxWorks підтримує протоколи маршрутизації RIPv1/RIPv2 (Routing Information Protocol), а також OSPF (Open Shortest Path First) версії 2. Протокол RIP входить в стандартну поставку VxWorks, OSPF поставляється як додатковий продукт. SNMP-агент для VxWorks підтримує протокол SNMP (Simple Network Management Protocol) як версії v1, так і v2c. MIB (Management Information Base) компілятор підтримує об'єкти MIB-II та розширення. STREAMS - стандартний інтерфейс для підключення переносних мережевих протоколів до операційних систем. У середовищі VxWorks можна інсталювати будь-який протокол, який має STREAMS-реалізацію: як стандартний (Novell SPX / IPX, Decnet, AppleTalk, SNA і т.п.), так і спеціалізований. VxWorks підтримує STREAMS версії UNIX System V.4.

Графічні пакети і вбудований Інтернет. Графічні програми для вбудованих комп'ютерів з ОСРВ VxWorks можуть бути розроблені як на мові С / С + +, так і на мовах Java і HTML. Для розробки графічних користувальницьких інтерфейсів (GUI) мовою C + + поставляється програмний продукт Zinc for VxWorks, для розробки на мові Java - PersonalJWorks і для розробки на мові HTML - HTMLWorks / eNavigator. Всі три GUI для VxWorks використовують один і той же універсальний API до графічної апаратури (графічному контролеру, фрейм-буферу і пристрою вводу), який називається UGL (Universal Graphics Library). UGL - це набір графічних примітивів 2D, драйвери популярних графічних контролерів і засоби розробки власних користувальницьких графічних драйверів. UGL входить до складу кожного GUI-продукту і поставляється в вихідних текстах.

Zinc for VxWorks - це C + + API, що надає широкий набір графічних об'єктів з вживаними користувачем параметрами. Для розробки GUI використовується Zinc Designer - WYSIWYG-редактор, який входить в комплект постачання. Графічний інтерфейс може бути розроблений на мові Java з використанням стандартного інструментарію pAWT (Abstract Windowing Toolkit), що входить до складу PersonalJWorks. Для розробки GUI використовується будь-який інструментарій розробки Java-додатків. Інтерфейс користувача може бути розроблений з використанням графічних можливостей мови HTML (фрейми, зображення, таблиці, форми) і динамічних можливостей JavaScript.htmlWorks - це інтерпретатор HTML / JavaScript-сторінок, які можуть знаходитися в постійній пам'яті або бути завантажені по мережі. Для розробки GUI використовується будь-який інструментарій web-дизайну. Якщо вбудований комп'ютер з HTML GUI повинен уміти виконувати web-серфінг, то спільно з HTMLWorks може бути використаний браузер для вбудованих додатків eNavigator.

Засоби побудови мультипроцесорних систем. VxWorks підтримує два види мультіпроцессінга: слабозв'язаних - через розподілені черги повідомлень і сільносвязаний - через об'єкти в поділюваної пам'яті. Слабозв'язаних мультіпроцессінг через розподілені черги повідомлень реалізований в бібліотеці VxFusion, яка є окремим продуктом. VxFusion застосовується для обміну між процесорами, що не мають загальної пам'яті (наприклад, між вузлами мережі). Сільносвязанний мультіпроцессінг через об'єкти в поділюваної пам'яті реалізований в бібліотеці VxMP, яка також є окремим продуктом. VxMP застосовується для обміну між процесорами, що мають загальну область пам'яті (наприклад, знаходяться на одній шині).

Засоби портування. Всі апаратно-залежні частини VxWorks винесені в окремі модулі для того, щоб розробник вбудовуваної комп'ютерної системи міг сам портувати VxWorks на свій нестандартний цільовий комп'ютер. Цей комплект конфігураційних і ініціалізаціонних модулів називається BSP (Board Support Package) і поставляється для стандартних комп'ютерів (VME-процесор, PC або Sparcstation) у вихідних текстах. Розробник нестандартного комп'ютера може взяти за зразок BSP найбільш близького за архітектурі стандартного комп'ютера і портувати VxWorks на свій комп'ютер шляхом розробки власного BSP за допомогою BSP Developer's Kit.

Проміжне ПЗ (middleware). Модель компонентних об'єктів COM (Component Object Model) та її розширення для розподілених систем DCOM (Distributed COM) є стандартними інтерфейсами обміну між додатками для Windows. VxDCOM - DCOM для операційної системи VxWorks - це перша реалізація моделі розподілених компонентних об'єктів для систем реального часу. Тепер немає необхідності в розробці спеціалізованих драйверів вводу / виводу при інтеграції нижнього і верхніх рівнів розподіленої системи управління. VxDCOM підтримує також OPC-інтерфейси (OLE for Process Control), що дозволяє розробляти OPC-сервери для вбудованих систем, що працюють під управлінням ОСРВ VxWorks.

Файлова система для флеш-пам'яті. Файлова система TrueFFS призначена для емуляції жорсткого диска, що працює під управлінням файлових систем VxWorks: DOS-FS і NFS (Network File System). TrueFFS підтримує стандарт PCMCIA FTL (Flash Translation Level) і підтримує PC-cards, MiniatureCards і мікросхеми флеш-пам'яті Intel 28F0xx, AMD 29F0xx, і Samsung 29Vxx000.

## 2. QNX Neutrino RTOS

Операційна система QNX Neutrino Realtime Operating System (RTOS) [QNXNeutrino] корпорації QNX Software Systems є мікроядерного операційною системою, яка забезпечує багатозадачність з пріоритетами. QNX Neutrino RTOS має клієнт-серверну архітектуру. У середовищі QNX Neutrino кожен драйвер, додаток, протокол і файлова система виконуються поза ядром, у захищеному адресному просторі. У разі збою будь-якого компонента він може автоматично перезапуск без впливу на інші компоненти або ядро. Хоча система QNX є конфігурується, тобто окремі модулі можна завантажувати статично або динамічно, не можна сказати, що вона використовує підхід, заснований на компонентах. Всі модулі покладаються на базове ядро і спроектовані таким чином, що не можуть використовуватися в інших середовищах.

QNX Neutrino RTOS складається з ядра, планувальника процесів (process manager) і розширених сервісів на рівні користувача. Як справжня мікроядерного операційна система, QNX Neutrino RTOS реалізує в ядрі ОС тільки найбільш фундаментальні сервіси, такі як передача повідомлень, сигнали, таймери, планування потоків, об'єкти синхронізації. Всі інші сервіси ОС, драйвери та програми виконуються як окремі процеси, які взаємодіють через синхронну передачу повідомлень.

Ядро QNX Neutrino RTOS виконується на рівні 0, керуючі програми і драйвери пристроїв виконуються на рівні 1 та 2, здійснюючи операції вводу / виводу. Програми виконуються на рівні 3.

Планувальник процесів будується на базисі ядра і забезпечує додаткову семантику рівня процесів, управління пам'яттю та шляхами доступу до файлів. Всі інші компоненти - файлові системи, набір протоколів, черги повідомлень, додатки - виконуються в захищеному адресному просторі і є розширеними сервісами. Взаємодія компонентів здійснюється через передачу повідомлень. Передача повідомлень грає роль віртуальної "програмної шини", яка дозволяє оперативно динамічно довантажувати і відвантажувати будь-який компонент. Як наслідок, будь-який модуль, навіть драйвер пристрою, може бути заміщений або перезапущений оперативно, для чого в більшості ОСРВ потрібно перезапустити системи. Повідомлення передаються прозоро через кордони процесора, забезпечуючи безшовний доступ до будь-якого ресурсу в мережі.

Володіючи які витісняють мікроядром і планувальником з пріоритетним обслуговуванням, QNX Neutrino RTOS здатна швидко і з високою передбачуваністю реагувати на події реального часу. Високопріоритетні потоки обробляють дедлайни своєчасно навіть при великій завантаженні системи (див. мал.2)

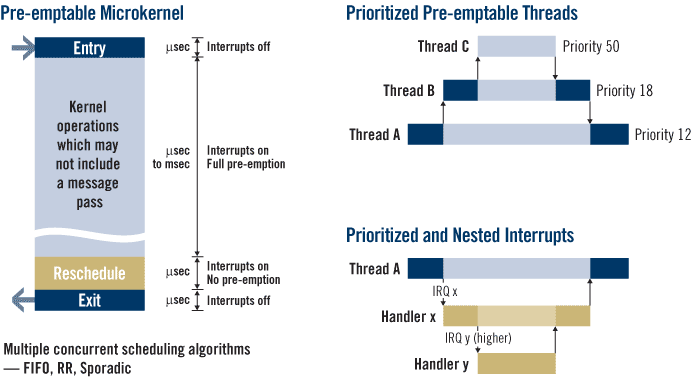


Рис.2. Продуктивність реального часу QNX Neutrino RTOS.

QNX Neutrino RTOS має малі часи обробки переривань, швидке перемикання контекстів. Інверсія пріоритетів долається за допомогою розподіленого успадкування пріоритетів. Спрощене моделювання активностей реального часу проводиться через синхронну передачу повідомлень. Вкладені переривання і фіксована верхня межа часу обробки переривання гарантують, що високопріоритетні переривання обробляються швидко з передбачуваним часом.

## 3. RTEMS

RTEMS (Real-Time Executive for Multiprocessor Systems) - це некомерційна операційна система реального часу для глибоко вбудованих систем [RTEMS]. Розробник системи компанія OAR (On-Line Applications Research Corporation, США). Система була створена на замовлення міністерства оборони США для використання в системах управління ракетними комплексами. Система розробляється для багатопроцесорних систем на основі відкритого вихідного коду на противагу аналогічним системам з закритим кодом. Система розрахована на платформи MS-Windows і Unix (GNU / Linux, FreeBSD, Solaris, MacOS X).

Ядро RTEMS забезпечує базову функціональність систем реального часу. У ці можливості входять

мультизадачність обробка;

робота в гомогенних і гетерогенних системах;

планування, кероване подіями, на основі пріоритетів;

планування з монотонною швидкістю;

взаємодію задач і синхронізація;

пріоритетне спадкування;

управління у відповідь перериванням;

розподіл динамічної пам'яті;

конфігурування системи для уповноважених користувачів;

переносимість на багато цільові платформи.

Ядро RTEMS відповідає за управління основною пам'яттю комп'ютера і віртуальною пам'яттю виконуваних процесів, за керування процесором і планування розподілу процесорних ресурсів між спільно виконуваними процесами, за управління зовнішніми пристроями і, нарешті, за забезпечення базових засобів синхронізації та взаємодії процесів. При цьому ядро використовує відповідні менеджери. До складу RTEMS входить набір наступних менеджерів: ініціалізації, завдань, переривань, годинника реального часу, таймер, семафорів, повідомлень, подій, сигналів, розділів, регіонів, двухпортової пам'яті, вводу / виводу, невиправних помилок, монотонною частоти, розширень користувача, багатопроцесорними. Прив'язка ОСРВ до апаратури проводиться за допомогою спеціальної бібліотеки підпрограм BSP (board support package) і спеціалізованих підпрограм для різних архітектур. До складу BSP входять програма ініціалізації апаратури і драйвери пристроїв. Підтримка в RTEMS мультипроцесорних систем дозволяє використовувати її для управління як однорідними, так і неоднорідними системами Ядро RTEMS автоматично враховує відмінності в архітектурі використовуваних процесорів, виконуючи у разі необхідності перестановку байтів і інші процедури. Це дозволяє здійснювати перехід на інше сімейство процесорів без значних змін системи.

ОСРВ RTEMS можна розглядати як набір компонентів, що забезпечують ряд базових сервісних функцій для програм користувача. Програмний інтерфейс програми складається з директив, розподілених по логічним розділами відповідних менеджерів. Функції, що використовуються декількома менеджерами, такі як розподіл процесорного часу, диспетчеризація і управління об'єктами, реалізовані в ядрі. Ядро містить також невеликий набір процедур, що залежать від типу використовуваного процесора: доступ до фізичної пам'яті, ініціалізація контролера переривань і периферійних пристроїв, специфічних для даного процесорного ядра, і т.д.

У ОСРВ RTEMS реалізуються наступні види міжпроцесорного взаємодії:

обмін даними між завданнями;

обмін даними між завданнями і програмами обробки переривань;

синхронізація між завданнями;

синхронізація між завданнями і програмами обробки переривань.

Функції, що дозволяють здійснювати ті чи інші види міжпроцесорного взаємодії, входять до більшості менеджерів RTEMS. Менеджери семафорів, повідомлень, подій, сигналів призначені виключно для здійснення міжпроцесорної взаємодії.

Менеджер семафорів. RTEMS підтримує стандартні виконавчі і семафори з лічильниками, що забезпечують синхронізацію і монопольний доступ до ресурсів.

Менеджер подій. Служить для синхронізації виконання завдань. Прапор події використовується завданням для того, щоб інформувати іншу задачу про виникнення певної події. Кожній задачі відповідають 32 прапори подій. Сукупність одного або більше прапорів називається набором подій. Одна задача може послати другий задачі набір подій, а також з'ясувати стан набору подій відповідною функцією.

Менеджер повідомлень. Служить для обміну між завданнями повідомленнями змінної довжини. Повідомлення передаються через черги типу FIFO ("першим прийшов, першим обслужений"). Є можливість посилки термінового повідомлення. Для кожної черги задається максимальна довжина повідомлення. Повідомлення можуть використовуватися для синхронізації завдань. Завдання може очікувати приходу певного повідомлення або перевіряти наявність повідомлення в черзі.

Менеджер сигналів. Використовується для асинхронного взаємодії між завданнями. Завдання може включати в себе процедуру обробки асинхронного сигналу, якій передається управління при отриманні сигналу. Прапор сигналу використовується завданням для того, щоб проінформувати іншу задачу про виникнення нештатної ситуації. Кожній задачі відповідають 32 прапори сигналів. Сукупність одного або більше прапорів називається набором сигналів.

Менеджер завдань. Забезпечує повний набір функцій для створення, видалення і управління завданнями. З точки зору RTEMS, завданням є найменша послідовність команд, яка може самостійно конкурувати за використання системних ресурсів. Кожній задачі відповідає блок контролю завдання TCB (Task Control Block). Цей блок є структурою, яка містить всю інформацію, що стосується виконання завдання. У процесі ініціалізації RTEMS виділяє TCB для кожного завдання, що є в системі. Елементи TCB змінюються відповідно до системними викликами, які виконуються додатком у відповідь на зовнішні запити. Блок TCB - це єдина внутрішня структура даних RTEMS, доступна додатком через додаткові процедури користувача. При перемиканні задач у TCB зберігається контекст завдання. При поверненні управління задачі її контекст відновлюється. При перезапуску завдання вихідний контекст завдання відновлюється відповідно зі стартовим контекстом, що зберігається в TCB. Завдання може знаходитися в одному з п'яти станів: виконання; готовність до виконання (управління може бути передано задачі); зупинка (завдання заблокована); сплячий режим (створена, але не запущена завдання); відсутність завдання (завдання не створена або видалена).

Ядро реального часу RTEMS підтримує 255 рівнів пріоритетів. Чим більше значення пріоритету, тим більше привілейованої є завдання. Кількість завдань, що мають однаковий пріоритет, не обмежена. Кожне завдання завжди має будь-якої рівень пріоритету, початкове значення якого присвоюється при створенні завдання і в подальшому може бути змінено. Режим виконання завдання визначається такими параметрами: витісняємість; обробка асинхронних запитів ASR (Asynchronous Signal Request); квантування часу; рівень переривання. Ці параметри використовуються для розподілу процесорного часу і зміни контексту завдання. Вони задаються користувачем при компіляції системи.

Параметр витісняємості визначає порядок передачі управління між завданнями. Якщо він включений, то завдання збереже контроль над процесором, поки вона перебуває в стані виконання, навіть якщо готова до виконання більш привілейована завдання. Якщо цей параметр вимкнено, то управління буде негайно передано задачі, що має більш високий пріоритет.

Параметр квантування часу визначає, як відбувається розподіл процесорного часу між завданнями з однаковим пріоритетом. Якщо він включений, то RTEMS обмежить час виконання завдання при наявності іншої задачі з таким же пріоритетом, готової до виконання. Час, що виділяється кожної такої задачі, визначається в таблиці конфігурації системи. Якщо квантування часу вимкнено, то завдання буде виконуватись до тих пір, поки не стане готова до виконання завдання з більш високим пріоритетом. Якщо параметр витісняємості вимкнено, параметр квантування часу не враховується.

Параметр обробки асинхронних сигналів (запитів) ASR визначає порядок обробки отриманих завданням сигналів (запитів). Якщо він включений, то послані задачі сигнали будуть оброблені, якщо вимкнено - сигнали будуть оброблені тільки після включення цього параметра. Цей параметр впливає тільки на завдання, що мають процедури обробки зовнішніх сигналів.

Параметр рівня переривання визначає, які переривання можуть оброблятися під час виконання завдання.

Менеджер ініціалізації. Відповідає за запуск і зупинку RTEMS. Ініціалізація RTEMS проводиться шляхом створення та запуску всіх ініціюючих завдань і ініціюючих процедур для кожного драйвера. У разі мультипроцесорної системи відбувається також ініціалізація механізмів міжпроцесорного взаємодії. Ініціюючих завдання відрізняються від решти завдань тим, що вони присутні в таблиці ініціюючих задач користувача і автоматично створюються RTEMS в процесі ініціалізації. Щоб ці задачі виконувалися до запуску решти завдань, вони повинні мати більш високий пріоритет. Після закінчення ініціалізації RTEMS не видаляє ініціюючих завдання, тому такі завдання повинні або самі видалити себе, або трансформуватися в "звичайну" завдання. У будь-якій системі повинна бути, як мінімум, одна ініціюючих завдання.

Менеджер переривань дозволяє швидко реагувати на переривання, забезпечуючи можливість "витіснення" завдання відразу після виходу з процедури обробки переривання. Менеджер переривань також дає програмі користувача можливість підключити процедуру обробки до відповідного вектора переривання. Коли надходить запит переривання, процесор передає його ядру RTEMS. При обслуговуванні запитів переривання RTEMS зберігає і відновлює вміст всіх регістрів, збереження яких не передбачене правилами мови С, а потім викликає налаштовувану процедуру обробки переривання. Для мінімізації часу, протягом якого не обслуговуються запити переривання рівного чи більш низького рівня, процедура обробки повинна виконувати лише мінімальний набір необхідних дій. Подальша обробка повинна здійснюватися програмою користувача. Менеджер переривань гарантує правильний розподіл процесорного часу між завданнями після завершення процедури обробки переривання. Системний виклик, зроблений з процедури обробки переривання, може перевести у стан готовності до виконання завдання з великим пріоритетом, ніж перервана завдання. Тому необхідно провести відкладену диспетчеризацію після завершення процедури обробки переривання. Виклик директив RTEMS з процедури обробки переривання не супроводжується диспетчеризацією.

Для правильного розподілу процесорного часу між завданнями повинно виконуватися така умова: всі процедури обробки переривань, які можуть бути перервані процедурами обробки переривань, що викликають директиви RTEMS з великим пріоритетом, повинні використовувати менеджер переривань. Якщо при обробці переривання надходить новий запит на переривання, його обробка відбувається відразу після завершення поточної процедури обробки. Відкладена диспетчеризація здійснюється тільки після того, як будуть обслужені всі запити. ОСРВ RTEMS підтримує 256 рівнів переривань, що транслюються в рівні переривань процесора.

При виконанні певних директив RTEMS може виникнути необхідність відключення обробки переривань, щоб забезпечити безперервне виконання критичних сегментів програми. Перед виконанням цих сегментів система RTEMS відключає всі маскіруємі переривання. Максимальний час вимикання переривань-різному для різних процесорів і вказується в документації RTEMS для відповідного процесора. Немаскіруємі переривання не відключаються, тому в процедурах їх обробки не повинні використовуватися директиви RTEMS.

Менеджер вводу / виводу. Забезпечує певний механізм доступу до драйверів пристроїв. Якщо в системі використовується цей менеджер, то в конфігураційної Таблиця повинна бути вказана адреса таблиці драйверів пристроїв, яка містить вхідні точки кожного драйвера. Драйвер може мати такі точки входу: ініціалізації, відкриття, закриття, читання, запису, контролю.

Менеджер доступу до пам'яті. Для роботи з пам'яттю служать менеджери розділів і регіонів. Розділ - це область пам'яті, що складається з буферів фіксованої довжини. Кожен з цих буферів може бути виділений для використання за допомогою директив менеджера розділів. Регіон - це область пам'яті змінної довжини, кратній розміру сторінки для даного регіону. Розділ представляє собою список буферів. При запиті на виділення буфера він виділяється з початку списку вільних буферів. Коли буфер звільняється, він поміщається в кінець цього списку. Регіон складається з блоків пам'яті різного розміру, який кратний розміру сторінки для даного регіону. Під час отримання запиту на виділення блоку пам'яті розмір запитаного блоку округляється до цілого кількості сторінок, і за наявності вільного блоку відповідного розміру цей блок виділяється. Менеджер доступу до пам'яті реалізує наступний набір функцій: створення, видалення, установка значень, звільнення, захоплення областей регіонів / розділів і буферів, що містяться в них. Для регіонів реалізується можливість додавання пам'яті.

Менеджер таймерів забезпечує роботу з таймерами: створення та видалення таймерів, доступ до таймерам, запуск підпрограм по події / сигналу від таймера. Цей менеджер може бути використаний для створення охоронного таймера.

Менеджер годин реального часу використовується для інформування користувача про поточну дату. Забезпечує також формування та обробку сигналів про закінчення мінімальних проміжків часу, які задаються на етапі конфігурування системи і рівні цілого числа мікросекунд.

RTEMS не підтримує динамічну завантаження додатків і модулів, тому сферою її застосування є вбудовувані системи, в яких не передбачається часта модифікація програмного забезпечення. ОСРВ RTEMS забезпечує досить слабку підтримку файлових систем, що обмежує область її можливого застосування в сфері систем централізованого збору та зберігання даних стандартними високорівневим засобами. На справжній момент RTEMS підтримує тільки файлові системи IMFS і TFTP, що явно недостатньо. Тому для створення на базі RTEMS файл-серверів потрібна розробка спеціального протоколу. Розуміючи цю проблему, розробники RTEMS ведуть активну роботу з реалізації систем підтримки широко використовуваних файлових систем (у першу чергу мережевих). У RTEMS фактично відсутні резидентні засоби відлагодження. Є тільки стандартні функції rtems\_panic і printf, які дозволяють виводити налагоджувальну інформацію на термінал у процесі роботи системи. Слід, однак, відзначити, що наявність потужних засобів крос-розробки робить цей недолік не дуже істотним.

## 4. ChorusOS

Операційна система ChorusOS - це розширювана вбудовувана ОС, широко застосовувана в телекомунікаційній індустрії. В даний час цей бренд розвивається і поширюється корпорацією Sun Microsystems [CHORUSOS]. Для компонування і розгортання ОС ChorusOS на конкретних телекомунікаційних платформах Sun Microsystems пропонує використовувати середовище розробки Sun Embedded Workshop. Корпорація Sun Microsystems представляє ОС ChorusOS як вбудовується основу для Sun'овской мережі, керованої сервісами (Sun's Service-Driven Network). У поєднанні з широким набором сервісів, повною інтеграцією ПЗ та апаратури, зручним адмініструванням і підтримкою Java-технології, яка присвячена потребам телекомунікації, ОС ChorusOS дає можливість ефективно розгортати нові можливості та програми, підтримуючи надійність і функціональність сучасних мереж.

ОС ChorusOS підтримує на одній апаратній платформі широкий набір телекомунікаційних протоколів, успадкованих додатків, додатків режиму реального часу і Java-технології.

ОС ChorusOS моделює три сорти додатків:

POSIX-процеси становлять більшість додатків ChorusOS; ці програми мають доступ до чисто POSIX API, декільком POSIX-подібним розширеним API і невеликого числа обмежених системних викликів мікроядра,

Актори ChorusOS - ці програми виконуються над мікроядром і обмежуються API мікроядра, актори включають драйвери, події підсистем і протокольні стеки,

Успадковані програми ChorusOS підтримуються для сумісності з додатками, розробленими для більш ранніх версій ChorusOS.

Архітектура ОС ChorusOS є багатошаровою, заснованої на компонентах (component-based). Мікроядро містить мінімальний набір компонентів, необхідних для функціонування ОС

kern - реалізує інтерфейс мікроядра і містить актор KERN, допоміжну бібліотеку і заголовні файли,

менеджер приватних даних (pd) реалізує інтерфейс між підсистемами мікроядра,

менеджер постійної пам'яті (pmm) реалізує інтерфейс постійної пам'яті,

core executive забезпечує істотну частину підтримки реального часу.

Компонент диспетчера ядра (core executive) забезпечує наступну функціональність

підтримка численних незалежних додатків,

підтримка користувацьких і системних додатків,

підтримка актора - одиниці модулярізаціі додатків,

підтримка одиниці виконання - потоку,

операції управління потоками,

управління Local Access Point (LAP),

сервіси управління винятковими ситуаціями,

мінімальний сервіс управління переривань.

У core executive відсутній управління такими сутностями, як синхронізація, планування, час, пам'ять. Політики керування цими поняттями забезпечуються додатковими компонентами, які вибираються користувачем в залежності від вимог апаратних і програмних засобів. Core executive завжди присутній у виконуваному примірнику ОС ChorusOS, інші компоненти конфігуруються і додаються по необхідності. Розмір резидентної частина ядра складає 10Kb.

Поняття "актор" в ChorusOS визначається як одиниця завантаження для програми. Воно також служить одиницею інкапсуляції для того, щоб зіставити всі системні ресурси, що використовуються додатком, і потоки, що виконуються всередині актора. Прикладами таких ресурсів є потоки, регіони пам'яті і кінцеві точки взаємодії.

Необов'язкові компоненти ОС ChorusOS 5.0 розбиваються відповідно до функціональністю:

Управління діяльністю (Actor management) включає підтримку розширення режиму користувача, динамічні бібліотеки, управління стиснутими файлами;

Планування (Scheduling) включає планування в стилі FIFO (first-in-first-out), різностильних планування (multi-class scheduling), циклічне планування (round-robin), планування в режимі реального часу;

Управління пам'яттю включає, крім розподілу пам'яті, підтримки апаратного захисту і підкачки, ще й статистику мікроядра, події системи Solaris, метрики операційної системи;

Працездатність (High Availability) включає гарячий рестарт, сторожовий таймер (Watchdog timer), чорний ящик, дамп системи;

Синхронізація потоків включає семафори, набори прапорів подій, мьютекс, монопольні блокування, що забезпечують відсутність інверсії пріоритетів;

Управління часом включає періодичні таймери, потоковий віртуальний таймер, дата і час, датчик реального часу, змінні оточення;

Взаємодія потоків включає незалежне від місцезнаходження взаємодія, підтримку віддаленого взаємодії, механізм взаємодії через поштові скриньки, синхронізацію між потоками, приватні дані потоку, а також такі засоби взаємодії системи POSIX, як семафори, сокети, потоки, таймери, черги повідомлень, об'єкти поділюваної пам'яті, сигнали реального часу;

Інструментальна підтримка включає системну журналізацію, реєстрацію помилок, підтримку профілювання і контрольних точок, моніторинг системи, налагодження системи, дамп ядра;

Підтримка мови C включає командний інтерпретатор на цільовому комп'ютері, віддалений shell;

Підтримка файлової системи включає іменовані канали, NFS-клієнт, NFS-сервер, файлові системи MS-DOS, PDE, / proc, UFS, ISO9000;

Управління введенням / виводом включає підтримку драйверів деяких пристроїв;

Мережева підтримка включає підтримку деяких мережевих протоколів.

Виділення управління пам'яттю в окремий необов'язковий компонент дозволяє легко адаптувати систему до різних апаратних платформ.

ОС ChorusOS 5.0 лежить в основі операційного середовища Solaris і підтримує такі цільові платформи:

UltraSPARC II (CP1500 і CP20x0),

Intel x86, Pentium,

Motorola PowerPC 750 і сімейство процесорів 74x0 (mpc7xx),

Motorola PowerQUICC I (mpc8xx) і PowerQUICC II (mpc8260) (мікроконтролери).

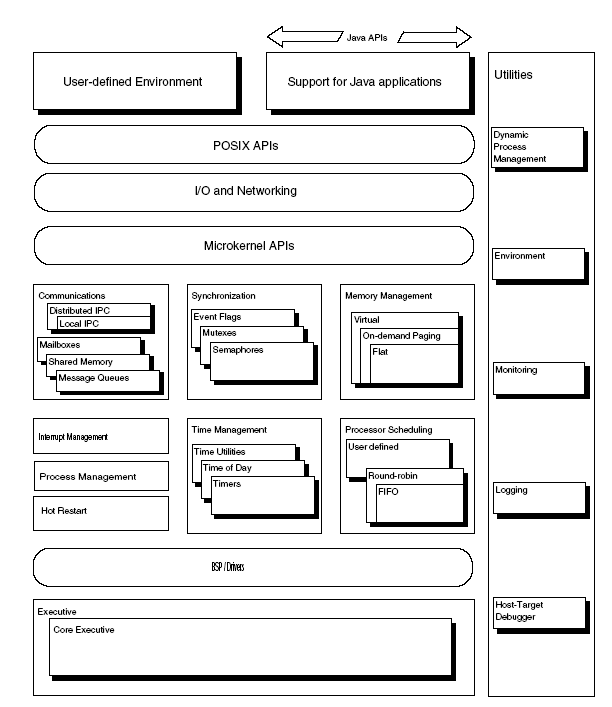


Рис.3. Архітектура ChorusOS.

## 5. Розширення реального часу для Windows NT

Windows NT проектувалася і, в основному, використовується як універсальна ОС. Однак на ринку систем реального часу чітко простежується тенденція використовувати Windows NT і її розширення в спеціалізованих системах. На це існує кілька причин:

Windows NT проектувалася відповідно до сучасних технологій побудови ОС,

програмний інтерфейс додатків (API) для Win32 став де-факто стандартом для програмістів,

графічний користувальницький інтерфейс (GUI) став настільки популярним, що інші ОС намагаються забезпечити схожий інтерфейс,

доступна велика кількість драйверів пристроїв,

доступні багато потужні інтегровані середовища розробки.

Сама по собі Windows NT не підходить для застосування в системах реального часу, оскільки в ній дуже мало пріоритетних рівнів, відсутній механізм успадкування пріоритетів. Для мінімізації часу обробки переривань (ISR) в Windows NT введена концепція відкладеного виклику процедури (DPC - deferred procedure call), пріоритет якої вище, ніж пріоритет для користувача і системних потоків, у той час як всі DPC мають однаковий пріоритет. Це призводить до того, що всі DPC ставляться в чергу FIFO, і DPC з високорівневим перериванням зможе виконатися тільки після того, як всі інші DPC, що стоять в черзі перед нею, будуть виконані. Такі ситуації ведуть до непередбачуваних часи відгуку, що несумісно з вимогами до ОСРВ. Управління пам'яттю в Windows NT засновано на механізмі віртуальної пам'яті. Це тягне за собою захист пам'яті, трансляцію адрес і підкачування, яка неприйнятна в ОСРВ.

## 5.1 RTX для Windows NT

Розширення реального часу RTX (Real Time Extension) для ОС Windows NT (розроблено корпорацією VenturСom) дозволяє створювати додатки для високошвидкісного керування з детермінованим часом реакції на зовнішні події [RTX].

RTX глибоко інтегроване в ядро Windows NT і для забезпечення необхідних функцій використовує сервіс Windows NT і API WIN32. Ядро реального часу (nucleus) інтегровано у ядро NT (kernel). Кожен процес RTX виконується як драйвер пристрою ядра NT, при цьому процеси не захищені один від одного. Така реалізація приводить до швидкого переключення контексту, але небезпечна з точки зору конфіденційності.

Розширення реального часу додають до Windows NT специфічну для реального часу функціональність.

Забезпечується можливість створювати процеси реального часу, керовані власним планувальником. Цей планувальник працює вже за правилами реального часу і використовує алгоритм витіснення за пріоритетами. Крім того, процеси реального часу мають перевагу перед стандартними процесами Win32, витісняючи їх. Процеси реального часу мають зовсім іншу, порівняно зі стандартними процесами Windows NT, ступінь надійності і специфічну функціональність.

Процеси реального часу і стандартні процеси Win32 мають засоби взаємодії один з одним.

Процеси реального часу мають свій власний програмний інтерфейс RTAPI, що реалізує розвинений набір засобів, характерний для програмних інтерфейсів (API) ОСРВ.

Додаток може використати як стандартні функції Win32, так і специфічні функції API реального часу (RTAPI), що дозволяє виділяти критичні ділянки коду додатків Windows NT і контролювати час та надійність їх виконання.

Є можливість контролю над працездатністю і часом реакції системи. Зависання стандартних програм Windows NT або крах системи не призводять до зависання додатків реального часу.

Надається можливість роботи зі швидкими годинником і таймерами високого дозволу.

Забезпечується можливість прямого доступу до пам'яті та фізичним пристроям.

RTX включає в себе наступні компоненти:

рівень абстракції апаратури HAL (Hardware Abstraction Layer) реального часу (Real-Time HAL). HAL є програмним компонентом найнижчого рівня при взаємодії драйверів ядра з апаратурою. Зокрема, саме на рівні HAL відбувається первинна обробка переривань від таймера,

підсистему реального часу RTSS (Real-Time Subsystem),

програмний інтерфейс розширень реального часу RTAPI (Real-Time Application Programming Interface). HAL реального часу підміняє стандартний HAL Windows NT.

Підсистема реального часу RTSS забезпечує виконання більшості функцій і керування ресурсами розширень реального часу. З точки зору реалізації, RTSS виглядає як драйвер Windows NT і виконується в режимі ядра. Це дозволяє досить простим способом влаштувати взаємодія між процесами реального часу і процесами Windows NT. RTSS забезпечує виконання функцій RTAPI і містить планувальник потоків реального часу з 128-ю фіксованими пріоритетами. RTSS містить також менеджер об'єктів, що надає уніфікованих механізмів використання системних ресурсів. У порівнянні з набором об'єктів Windows NT, додані такі об'єкти, як таймери і обробники переривань.

Робота з перериваннями Real-Time HAL. Перехоплюючи апаратні переривання, Real-Time HAL розрізняє переривання, пов'язані з обробникам реального часу і обробникам Windows NT. Переривання, які повинні оброблятися драйверами Windows NT, відправляються за стандартною ланцюжку. При цьому Real-Time HAL стежить за тим, щоб переривання не маскувалися драйверами Windows NT більш ніж на 5 мкс, виключаючи можливість пропуску критичного події.

Швидкі годинник і таймерні служби. Для вимірювання часових інтервалів або для генерації переривань Real-Time HAL дозволяє працювати з тикер, дозвіл якого 1 мкс. Системний таймер синхронізований з тикер, і може працювати з періодом 100 мкс або швидше, забезпечуючи роботу як стандартних таймерних сервісів, так і додаткових, що входять до складу підсистеми реального часу.

Підтримка підсистеми реального часу (RTSS). Крім перерахованих вище функцій (переривання і таймери), Real-Time HAL містить підтримку функціонування всієї підсистеми реального часу. Так, на основі переривань від таймера будується диспетчер процесів реального часу. Real-Time HAL відповідає також за виконання функцій вводу-виводу підсистеми реального часу і ін

Програмний інтерфейс реального часу RTAPI є розширенням Win32 і містить, перш за все, набір функцій, необхідних для керування пристроями. RTAPI реалізований у двох видах - як підмножина викликів підсистеми реального часу (RTSS) і як динамічна бібліотека (DLL), яка може викликатися з Win32-додатків. RTAPI містить наступні групи функцій:

управління процесами і потоками - надає Win32-сумісний інтерфейс для управління, створення, зміни пріоритетів, профілювання і завершення потоків реального часу,

управління об'єктами RTSS - надає можливості уніфікованого управління об'єктами RTSS (створення, закриття, доступ). Об'єктами RTSS є: таймери, обробники прерії-жень і виняткових ситуацій (startup, shutdown, blue screen), потоки, процеси, семафори, мьютекс (mutex), колективна пам'ять, поштові скриньки, консольний і файловий ввід-висновок, регістри.

Взаємодія між процесами. У RTAPI використовуються семафори, мьютекс і колективна пам'ять для взаємодії як потоків реального часу між собою, так і для їх взаємодії з процесами WIN32.

Управління пам'яттю дозволяє фіксувати програми в пам'яті, забороняючи їх вивантаження в файл підкачки.

Доступ до фізичної пам'яті: програма користувача отримує можливість доступу до даних по фізичних адресами пам'яті.

Управління переривань дозволяє призначати і забороняти обробники переривань, дозволяти і забороняти переривання.

Управління годинником і таймерами дозволяє створювати, видаляти, скасовувати, ініціалізувати таймери, призначати обробники переривань.

Керування вводом-висновком RTAPI надає два способи керування пристроями введення-виведення. По-перше, програми користувача отримують можливість безпосереднього доступу до адрес портів введення-виведення, що дозволяє програмувати роботу пристроїв напряму. Крім того, зовнішній пристрій може управлятися спеціальними (легко розроблюваними) драйверами, для роботи з якими RTAPI надає спеціальний інтерфейс.

## 5.2 INtime

Система INtime є розширенням реального часу Windows, яке було розроблене корпорацією Radisys Corporation, а в даний час підтримується корпорацією TenAsys [INTIME].

INtime комбінує можливості ОСРВ жорсткого реального часу зі стандартними ОС Windows, включаючи Windows XP, Windows XP Embedded, Windows 2000, Windows NT і Windows NT Embedded, не вимагаючи додаткової апаратури. INtime спеціально розроблена під архітектуру процесора x86. Програми реального часу і не реального часу виконуються на різних віртуальних машинах на одному комп'ютері (див. рис.4).

INtime, на відміну від RTX, слабо пов'язана з NT. Архітектура INtime заснована на механізмі апаратного обслуговування завдань (hardware tasking), яке забезпечується процесором Intel. Виходить, що два ядра виконуються на одній апаратурі. Оскільки вони поділяють одну апаратуру, потрібні були деякі модифікації NT HAL. Такий підхід дозволяє захистити і відокремити середовище виконання і область пам'яті від Windows. Всередині INtime кожен процес програми має своє власний адресний простір. Крім того, ядро і програми виконуються на різних рівнях пріоритетних, що дозволяє захистити їх один від одного.

INtime показує передбачувану поведінку, однак її складна архітектура не дозволяє досягти системі гарній продуктивності. Через сегментаційного обмежень INtime підходить не для всіх систем реального часу.

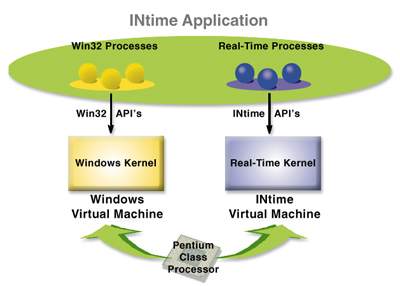


Рис.4. Структура INtime.

## 5.2.1 Microsoft Windows Embedded

Операційні системи Microsoft Windows Embedded для вбудованих систем мають два різновиди відповідно до версіями ОС Windows - NT і XP [MSEmb]. Версії систем Embedded корпорації Microsoft складаються з численних конфігурованих частин, які дозволяють легко маніпулювати набором встановленого програмного забезпечення.

Windows NT Embedded використовує технічні ресурси Windows NT і дозволяє розробляти додатки, які можуть бути легко інтегровані в існуючу інформаційну інфраструктуру.

Набір засобів розробки - Target Designer і Component Designer - дозволяє OEM (original equipment manufacturer) виробникам конфігурувати та створювати операційну систему для конкретної апаратної платформи. Windows NT Embedded володіє специфічними компонентами для створення вбудованих систем, які дозволяють працювати в системах без відеоадаптера, здійснювати завантаження і роботу накопичувачів в режимі "тільки читання", виконувати віддалене адміністрування і надають додаткові засоби обробки помилок і відновлення. Windows NT Embedded дає можливість створювати пристрої, з якими працювати так само просто, як і зі стандартними ПК на основі Windows, та управляти цими новими пристроями на основі існуючих професійних продуктів, таких як Microsoft Systems Management Сервер, HP OpenView, IBM Tivoli, CA Unicenter TNG, та ін

Розробник вбудованих систем застосовує для конфігурування ОС Target Designer, використовуючи готовий двійковий код Windows NT, додаткові компоненти для вбудовування і додаткові додатки. У разі необхідності, для створення нових компонентів, що не входять до складу продукту (наприклад, драйверів пристроїв, додатків та ін), може використовуватися Component Designer. Новостворені нові компоненти можуть бути імпортовані в Target Designer і включені до складу цільової ОС. Після конфігурування ОС з допомогою Target Designer відбувається перевірка взаємозв'язків компонентів і будується образ системи, готовий до завантаження і виконання на цільовій системі.

Windows XP Embedded налічує до 10000 окремих компонентів, а в Windows NT Embedded їх було трохи більше 300. Основною відмінною рисою Windows XP Embedded є чітке розмежування компонентів системи, що дозволяє розробникам вбудованого набору функцій при створенні образу системи включати тільки необхідні файли і максимально скоротити розмір результуючої системи. Цими компонентами служать окремі частини системи Windows XP Professional.

Компоненти Windows XP Embedded представлені сервісами, додатками, бібліотеками і драйверами - розробнику потрібно настроїти необхідний набір функцій і зібрати з компонентів необхідну конфігурацію в образ середовища виконання (runtime image). Всі опції конфігурації зібрані воєдино в базу даних компонентів. Розробник має до неї доступ і може її редагувати за допомогою спеціального інструменту - Component Database Manager.

Для кожного компонента в процесі створення визначається ряд параметрів:

платформа, на якій буде виконуватися даний компонент (визначає порядок компіляції та складання);

опис і схема підключення компонента;

список асоційованих ресурсів, таких як файли і ключі реєстру;

залежно компонента від інших компонентів (наприклад, від DirectX або NET runtime);

покажчик на сховище файлів (найчастіше це просто локальний каталог, але може бути і мережевим ресурсом);

приналежність до групи для спрощення звернення відразу до декількох компонентів як до цілого.

Сама база даних управляється СУБД MS SQL Server і може бути розташована як локально, на комп'ютері розробника, так і на сервері.

## 6. TinyOS

Розробка операційної системи TinyOS [HSW00] пов'язана з появою нової концепції бездротового зв'язку - Motes. Motes (у перекладі з англійської - порошинки, смітинки) - це реалізація ідеї "smart-dust" ("розпорошеної розумності"), запропонованої оборонним агентством Darpa (Defense Advanced Research Projects Agency), зокрема, для відстеження пересувань противника.

Motes розроблені в Каліфорнійському університеті в Берклі спільно з Intel, і в даний час ведуться випробування цих самоорганізуються мереж, побудованих на основі відкритих технологій Intel Mote та програмного забезпечення TinyOS, TinyDB.

Розумні сенсори Motes, розподілені в просторі, можуть самостійно зв'язуватися один з одним, утворюючи розподілену бездротову інформаційну мережу. "Порошинка розуму" складається з 8-бітового мікроконтролера сімейства Amtel AVR, прийомопередаюче інтегрального модуля TR1000 і двох мікросхем середнього ступеня інтеграції - енергонезалежної пам'яті і додаткового завантажувального мікроконтролера, що дозволяє по радіоканалу оновлювати ПО центрального процесора - AVR.

"Smart-dust" створювалася для динамічних, що змінюються як в просторі, так і в часі мереж - для тієї області, в якій абсолютно незастосовні ні традиційні алгоритми управління, ні відпрацьовані принципи маршрутизації, ні архітектурні рішення, що лежать в основі традиційного системного ПЗ. Прагнення конструкторів зробити її якомога компактнішою (у перспективі - 1 мм 3) тягне за собою низку істотних обмежень, в першу чергу енергетичних. Обмежені обчислювальні ресурси і динамічний характер мережі призводять до того, що функціональність "пилинки" треба час від часу змінювати, що може бути досягнуто тільки одним способом - передачею по радіоканалу потрібного ПЗ. З іншого боку, енергетична дорожнеча передачі інформації вимагає надкомпактний подання переданого коду, в іншому випадку "пилинки" просто не будуть працювати з-за швидкого виснаження крихітних автономних джерел живлення.

При проектуванні TinyOS основними вимогами були досягнення енергетичної ефективності і створення високого рівня абстракції системних викликів для спрощення розробки програм. Ця система володіє всіма відмінними рисами розвиненої ОС - в першу чергу, вкрай простий, але достатньо розвиненою компонентної моделлю. Однак специфіка призначення цієї компонентної моделі істотно відрізняється від традиційних розробок, оскільки головною метою компонентного є не полегшення підбору інтерфейсів, які відповідають вимогам запитуючої компонента, а забезпечення розвинених і надійних механізмів паралельного виконання завдань в умовах вкрай обмежених ресурсів.

Вищеописані причини призвели розробників TinyOS до вибору подієвої моделі, яка дозволяє управляти високим ступенем паралельності виконання завдань в обмеженому просторі пам'яті. Підхід до управління багатопоточності, заснований на стек, зажадав б значно більших ресурсів пам'яті, ніж передбачалося в даному проекті. Для кожного контексту виконання потрібно було б виділення пам'яті для найгіршого варіанту, або потрібно було б застосувати будь-якої занадто витончений і складний метод управління пам'яттю.

Архітектура TinyOS об'єднує таке звичну складову, як планувальник завдань (scheduler), і оригінальне поняття - компонентний граф. Термін "компонент" тут одночасно і відповідає загальноприйнятому розумінню, і суттєво розширює його. Так, інтерфейс компонента TinyOS складається з двох множин - верхнього, що надається цим компонентом, і нижнього, необхідного для його функціонування. Кожне з цих множин містить описи команд і подій - синхронних і асинхронних процесів.

Відповідно до опису системи, компонент має 4 взаємопов'язані частини - набір команд, набір обробників подій, інкапсулювання фрейм фіксованого розміру і пучок простих потоків. Потоки, команди і обробники подій виконуються в контексті даного фрейму і впливають на його стан. Крім того, кожен компонент декларує команди, які він використовує, і події, про які він сигналізує. Ці декларації використовуються при компонуванні для конфігурації системи, яка налаштована на певний клас додатків. Процес композиції створює шари компонентів, де кожен більш високий рівень видає команди до нижчого рівня, а нижчележачий рівень звертається до більш високого за допомогою сигналів, що розуміється в даній системі як події. Апаратне забезпечення є найнижчим шаром компонентів.

Написана TinyOS з використанням структурованого підмножини мови C. Використання статичного розподілу пам'яті дозволяє визначати вимоги до пам'яті на рівні компіляції та уникати накладних витрат, пов'язаних з динамічним розподілом. Крім того, цей підхід дозволяє скорочувати час виконання завдяки статичному розміщення змінних під час компіляції замість доступу до них за вказівником під час виконання.

Команди є неблокіруємими запитами до компонентів нижчого шару. Команда зберігає необхідні параметри в своєму фреймі і може ініціювати потік для його подальшого виконання. Щоб не виникало невизначених затримок, час відповіді від викликаної команди не повинно перевищувати заданого інтервалу часу, при цьому команда повинна повернути статус, який вказує, успішно вона завершилася чи ні. Команда не може подавати сигнали про події.

Обробники подій прямо або побічно мають справу з апаратними подіями. Самий нижній шар компонентів містить обробники, безпосередньо пов'язані з апаратними перериваннями. Оброблювач подій може покласти інформацію в свій фрейм, запустити потоки, подати сигнал вищого рівня про події або викликати команди нижчого шару. Апаратне подія ініціює фонтан обробки, яка поширюється вгору по рівнях через події і може повернутися вниз через команди. Щоб уникнути циклів у ланцюжку команд / подій, команди не можуть подавати сигнали про події. Як команди, так і події призначені для виконання невеликий, суворо фіксованій порції обробки, яка виникає всередині контексту що виконується потоку.

Основна робота покладається на потоки. Потоки в TinyOS є атомарними і, на відміну від потоків в інших ОС, виконуються аж до свого завершення, хоча вони і можуть бути витіснені подіями. Потоки можуть викликати команди нижчого рівня, сигналізувати про події більш високому рівню і планувати інші потоки усередині компонента. Семантика потоку "виконання аж до завершення" дозволяє мати один стек, який виділяється виконує потоку, що дуже істотно в системах з обмеженою пам'яттю. Потоки дають можливість симулювати паралельну обробку усередині кожного компонента, тому що вони виконуються асинхронно по відношенню до подій. Однак потоки не повинні блокуватися або простоювати в очікуванні, тому в таких випадках вони будуть перешкоджати розвитку обробки в інших компонентах. Пучки потоків забезпечують засіб для вбудовування довільних обчислювальних обробок у модель, керовану подіями.

У системі передбачена також окрема абстракція завдання, що розуміється як тривалий обчислювальний процес. Взаємовідношення між поняттями "команда" і "завдання" наступне: команда - це атомарна складова завдання. Команда ставиться в чергу на виконання планувальника, потім вона виконується і може бути тимчасово перервана обробкою події.

Планувальник працює за принципом черги FIFO, тобто для передачі керування наступної задачі потрібно повне завершення попередньої задачі. Однак, в залежності від вимог програми, можуть використовуватися і більш складні механізми планування, засновані на пріоритетах або на дедлайн. Ключовим моментом є те, що планувальник орієнтований на енергозбереження: процесор засинає, якщо чергу планувальника порожня, а периферійні пристрої працюють, і кожне з них може розбудити систему. Коли черга стає порожній, новий потік може бути запущений на виконання тільки в результаті якого-небудь події, яка може виникнути тільки в апаратних пристроях. Планувальник має вкрай малі розміри - всього 178 байт, дані планувальника займають лише 16 байтів.

У TinyOS повністю відсутні механізми блокування виконання, що означає необхідність введення індикації завершення тривалої операції відповідним асинхронним подією. Традиційні прийоми побудови ОС реального часу і звичні відпрацьовані архітектурні рішення тут виявилися незастосовні. У результаті вся ОС та її компоненти побудовані за принципом кінцевих автоматів - переходів зі стану в стан. Отже, TinyOS складається з набору компонентів (кожен розміром приблизно 200 байт), з яких розробники збирають систему для кожного конкретного сенсора. Для компонування системи з набору компонентів, які статично компонуються з ядром, використовується спеціальний описовий мову. Після проведення компонування модифікація системи не можлива.

Для забезпечення динамічності під час виконання була розроблена віртуальна машина, яка є надбудовою над ОС TinyOS. Код для віртуальної машини можна завантажити в систему під час виконання. Для роботи цієї віртуальної машини необхідні 600 байт оперативної пам'яті й менше 8 KB пам'яті команд 8-бітового мікроконтролера. Програми віртуальної машини представляються 8-бітовими інструкціями всього трьох типів, що об'єднуються в "капсули" - атомарні послідовності не більш ніж двадцяти чотирьох інструкцій. Ієрархічна структура мережі виходить автоматично завдяки тому, що всі сенсори слідують простим правилам, закладеним у TinyOS. Правила ці, наприклад, визначають спосіб пошуку найкоротшого шляху до найближчого стаціонарного вузла, а вже залежно від того, де і як розташовані сенсори, мережа приймає звичну для системних адміністраторів древообразну форму. У TinyOS враховується також і те, що деякі види сенсорів можуть працювати від сонячних батарей чи інших джерел енергії, що залежать від погоди, тому при втраті зв'язку з найближчим вузлом мережі відбувається зміна маршруту, по якому пересилаються пакети.

## 7. OSEK / VDX

Як уже згадувалося в розділі про стандарти, OSEK / VDX є комбінацією стандартів комп'ютерних систем реального часу, розроблених консорціумами OSEK і VDX для автомобільної промисловості. У даній роботі розглядається тільки стандарт OSEK, що стосується архітектури операційної системи.

ОС OSEK оперує такими об'єктами, як завдання, події, ресурси. Крім того, забезпечуються такі можливості, як управління помилками та засоби для користувача функцій відстеження змін у стані системи.

ОС OSEK забезпечує певний набір інтерфейсів для користувача. Інтерфейси використовуються сутностями, що конкурують за центральний процесор. ОС OSEK оперує двома типами таких сутностей - завдання і переривання - і визначає три рівні обробки - рівень переривань, логічний рівень планувальника і рівень завдань. Завдання вибираються на виконання відповідно до присвоєними їм пріоритетами.

Завдання в ОС OSEK може бути

базової або розширеної,

витісняється або невитискаючої.

Головна відмінність між базовою і розширеної завданнями полягає в тому, чи може завдання впасти в стан очікування (в якому вона чекає на появу події). Тільки розширена задача може очікувати події. Витісняється завдання може бути витіснена завданням більш високого пріоритету або перервана перериванням. Невитискаючої завдання може бути витіснена тільки за допомогою переривання (коли переривання не заборонені).

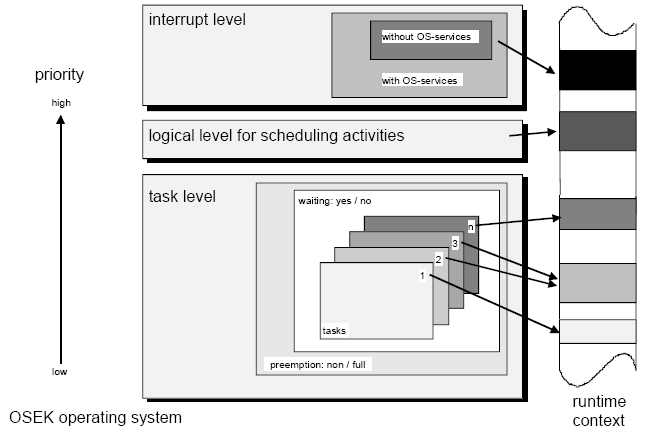


Рис.5. Рівні обробки в ОС OSEK.

Концепція двох типів завдань зажадала введення нового поняття - клас відповідності (conformance class) для опису своєрідною реалізації ОС OSEK і системних сервісів. Визначаються чотири класи відповідності - два для базового відповідності (BCC1 і BCC2 - Basic conformance Classes 1 і 2) і два для розширеного (ECC1 і ECC2 - Extended Conformance Classes 1 і 2). Реалізації, які відповідають базовим класам, вимагають використання тільки базових завдань, у той час як для розширених класів потрібні як розширені, так і базові завдання. Числа 1 і 2 в іменах класів вказують кількість запитів на завдання для базових завдань і кількість завдань на пріоритет для всіх завдань. Таким чином, в BCC1 і ECC1 є тільки одне завдання на пріоритет, і базові завдання можуть бути запитані тільки один раз. У BCC2 і ECC2 допускається множинність завдань на пріоритет і множинне запрашіваніе базових завдань.

Кожна задача повинна знаходитися в одному з чотирьох станів

Виконуються - тільки одне завдання може бути в цьому стані,

Готова до виконання - планувальник може вибрати її на виконання на підставі пріоритетів і правил витіснення,

Стані очікування - завдання чекає на появу події,

Призупинена - завдання в пасивному стані і чекає активації.

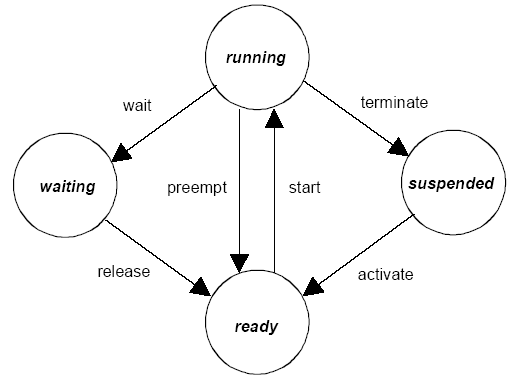


Рис.6. Модель станів завдання в ОС OSEK.

Кожне завдання має пріоритет. Стандарт ОС OSEK не обмежує максимальну кількість пріоритетів - це визначає реалізація.

ОС OSEK визначає два рівні програм управління переривань, які розрізняються можливостями виклику системних сервісів. Переривання рівня 1 виконуються незалежно від ОС дуже швидко. Рівень 2 забезпечує виконання функцій додатків, які містять виклики ОС.

Події в ОС OSEK використовуються для синхронізації різних завдань. Події є власністю завдань. Будь-яка задача, в тому числі і базова, може встановити подія, і тільки власник події може очікувати або зняти його.

Управління ресурсами забезпечує доступ до ресурсів, що розділяються, таким як пам'ять, апаратура і т.п. Планувальник також вважається спеціальним ресурсом, який може бути захоплений завданнями. Щоб уникнути інверсії пріоритетів і тупикових ситуацій, OSEK застосовує стельовий протокол пріоритетів. Згідно з цим протоколом задачі, що захопила ресурс, тимчасово підвищується пріоритет, і, таким чином, ніякі інші завдання, які звертаються до даного ресурсу, не зможуть виконуватись до тих пір, поки ресурс залишається захопленим. Проте, всі завдання з більш високим пріоритетом, ніж пріоритет завдання, яке захопило ресурс, все ще можуть виконуватися.

Аварійні сигнали і лічильники в OSEK використовуються для синхронізації активації завдань з повторюваними подіями. Аварійний сигнал статично присвоюється лічильнику, задачі і дії. Вплив може або активувати завдання, або встановити подія. Лічильники оперують тактами і можуть становити час, кількість прийнятих імпульсів і т.п. Кожна реалізація забезпечує один часовий лічильник, який використовується для планування періодичних подій. Всі інші лічильники управляються через API, є специфічними для конкретної реалізації і не можуть бути які переносяться.

У OSEK існує два типи аварійних сигналів: циклічні і одинарні. Циклічні аварійні сигнали застосовуються для диспетчеризації завдання, яка повинна запускатися періодично. Лічильник аварійного сигналу може бути встановлений в відносне або абсолютне значення. Параметри циклу і значення лічильника можуть встановлювати заново динамічно. ОС OSEK забезпечує мінімальні засоби для керування помилками часу виконання. Однак є можливість додаткового управління помилками під час розробки завдяки розширеній функціональності повернення управління. Причина такого рішення полягає в тому, що після того, як продукт запущений у виробництво, більшість можливих помилок може бути виявлено під час тестування (такі як "невірний ідентифікатор завдання", "ресурс зайнятий", "непередбачений виклик з рівня переривань" і т.д.). Під час виконання більшість системних сервісів не повертає коди помилок, але деякі сервіси, такі як аварійні сигнали, які можуть стартувати і зупинятися динамічно, повертають код помилки, якщо даний аварійний сигнал уже використовувався. ОС OSEK визначає два типи помилок - помилки програми та фатальні помилки. При помилку програми, коли програма намагається виконати несанкціоновану операцію (наприклад, активізувати неіснуючу завдання), цілісність внутрішніх даних все ще зберігається. Фатальні помилки виникають, якщо ОС виявляє порушення цілісності внутрішніх даних. При виявленні таких помилок викликається сервіс завершення роботи ОС.

## 8. OSE RTOS

Операційна система реального часу OSE RTOS, розроблена в корпорації ENEA, має ядро з пріоритетним плануванням [OSERTOS]. Це ядро сильно оптимізовано для забезпечення високої продуктивності і досить компактно для використання у вбудованих системах. OSE має архітектуру, керовану повідомленнями, з простими системними викликами. Передача повідомлень в OSE служить концептуальним шлюзом в розподілених багатопроцесорних вбудованих системах. Завдання посилають повідомлення один одному напряму через ОС без підтримки черг, поштових скриньок або інших проміжних механізмів. OSE RTOS підтримує підкачування, дублювання, динамічне оновлення коду і багато комунікаційні протоколи.

OSE RTOS пропонує три варіанти ядра, побудовані за одним принципом. OSE Epsilon - для глибоко вбудовуваної і SoC (system-on-chip) розробки. OSEck - компактне ядро для DSP. OSE Link Handler - для численних змішаних CPU / DSP проектів. Всі вони підтримують дуже маленька кількість системних викликів - від шести до восьми.

Архітектура OSE RTOS заснована на багатошарової моделі (рис.7).

Одиницею виконання в OSE RTOS є процес. Процеси можуть бути згруповані в блок, який може мати власний пул пам'яті. У ядрі OSE RTOS адресний простір належить сегменту, який може включати один або більше блоків. Відображення блоків у сегменти і відображення пулів у регіони дає можливість досягти повного захисту пам'яті та ізоляції програми. Блоки й пули можуть розміщуватися в одному або декількох сегментах.

OSE RTOS оперує різними типами і категоріями процесів.

Типи процесів:

Процеси переривань виникають у відповідь на апаратні або програмні переривання, виконуються до кінця, мають найвищий пріоритет і такий же контекст, як і всі інші процеси,

таймерні процеси переривання аналогічні процесам переривань, за винятком того, що вони передбачаються планувальником періодично відповідно до вказаним періодом часу,

пріоритетні процеси є найпоширенішими процесами в OSE RTOS і виконуються до тих пір, поки не будуть витіснені процесом переривання або процесом з більш високим пріоритетом,

фонові процеси виконуються строго в режимі циклічного обслуговування з квантуванням часу на пріоритетному рівні, який знаходиться нижче всіх пріоритетних процесів.

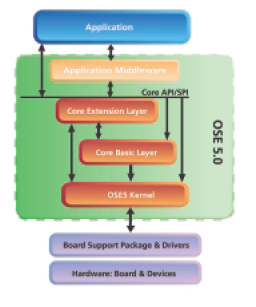


Рис.7. Багатошарова архітектура OSE RTOS.

Під категоріями процесів у OSE RTOS розуміється поділ процесів на динамічні та статичні. Статичні процеси створюються ядром, коли система стартує, і існують на всьому протязі існування системи. Динамічні процеси створюються і знищуються під час виконання.

Джерелом потенційних можливостей OSE RTOS є механізм прямої передачі повідомлень. Повідомлення, надіслане одним процесом іншому, містить ідентифікатор, адреси відправника і одержувача і дані. Як тільки повідомлення надіслано, відправник вже не має до нього доступу, тобто власність повідомлення ніколи не розділяється. Це важлива властивість виключає конфлікти доступу до пам'яті. Пряма передача повідомлень концептуально більш проста, ніж стандартна непряма модель, а унікальна розробка такої передачі виявилася надзвичайно ефективною.

## 9. Contiki

Операційна система Contiki [DGV04] розроблена в Швеції (Swedish Institute of Computer Science) для систем з обмеженою пам'яттю. Система Contiki дозволяє динамічно завантажувати і відвантажувати програми та сервіси. З метою мінімізації розмірів операційної системи було спроектовано ядро Contiki, яке засноване на моделі управління подіями [HSW00].

У традиційних системах, керованих подіями, процеси моделюються як обробники подій, які виконуються до завершення. Оскільки обробник подій не може бути заблокований, всі процеси можуть використовувати один і той же стек, розділяючи дефіцитні ресурси пам'яті. До того ж не потрібні механізми блокування, тому що два обробника подій ніколи не виконуються паралельно. В ОС, керованої подіями, довгі обробки монополізують центральний процесор, не даючи можливості реагувати на зовнішні події відбуваються. Однак, якщо ОС забезпечена механізмом багатопотокового обробки з перериваннями, цей недолік згладжується, що і зроблено в Contiki.

Багатопотоковий режим з пріоритетами в системі Contiki реалізований з допомогою бібліотеки додатків, які виконуються над ядром, керованим подіями. Додатки, що забезпечують багатопоточну обробку, компонуються з виконуючим додатком у міру необхідності, тобто якщо воно явно вимагає багатопотокового моделі обчислень. Виконувати система Contiki розділяється на дві частини - серцевину (core) та завантажені програми. Серцевина (core) складається з власне ядра (kernel), базових сервісів і фрагментів бібліотек підтримки, в тому числі мовної підтримки часу виконання. Колективна функціональність реалізується через сервіси як деяка форма спільних бібліотек. Ці сервіси можна оновлювати або заміщати динамічно незалежно один від одного під час виконання, що, на думку розробників, веде до гнучку структуру системи.

Реалізація Contiki показала, що багатопотокова обробка з пріоритетами необов'язково повинна бути захована на самий нижній пріоритетний рівень ядра, а може бути реалізована як бібліотека додатків над ядром, керованим подіями. Такий підхід дозволяє виконувати потокові програми над ядром без накладних витрат рентабельності або численних стеків у всіх частинах системи.

Системи, керовані подіями, мають свої проблеми. Модель програмування, керована станами, складна для програмістів. До того ж не всі програми укладаються в кінцево-автоматну модель.

Contiki не підтримує жодних механізмів захисту, тому що апаратура, для якої вона проектувалася, не підтримує захист пам'яті.

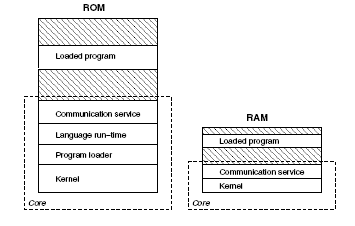


Рис.8. Серцевина Contiki та завантажені програми.

Що стосується архітектури ядра ОС Contiki, те ядро цієї системи складається з полегшеного планувальника, який здійснює диспетчеризацію подій для виконуються процесів і періодично викликає обробники опитування процесів. Виконання програми перемикається або у відповідності з подіями, регульованими ядром, або через механізм опитування. Якщо для обробки був обраний обробник події, ядро не перериває його роботу до тих пір, поки він не завершиться. Однак обробники подій можуть використовувати внутрішні механізми для виконання переривання. Ядро підтримує два види подій - асинхронні та синхронні. Асинхронні події є деякою формою відкладеного виклику процедури - асинхронні події ядро ставить в чергу, і вони направляються цільовим процесу якийсь час опісля. Синхронні події обробляються майже так само як асинхронні, тільки направляються цільовим процесу відразу. Управління повертається посилаєш процесу тільки після того, як цільовий процес завершив обробку події. Це можна розглядати як виклик процедури всередині процесу.

Contiki написана на мові C і адаптована для ряду мікроконтролерних архітектур, включаючи Texas Instruments MSP430 і Atmel AVR, а також для платформи ESB.

## 10. pSOS

ОСРВ pSOS була розроблена корпорацією Integrated Systems. В даний час вона належить корпорації WindRiver [PSOS], яка її купила, мабуть, для того, щоб вона не заважала на ринку збуту ОСРВ.

Ім'я pSOSsystem присвоєно операційній системі, ім'я pSOS + - її ядра. PRISM + - це інтегроване середовище розробки для створення додатків.

pSOS + - це маленьке ядро вбудованих додатків, що представляє собою якийсь варіант клієнт-серверної архітектури. Однак воно не має протоколу взаємодії, заснованого на повідомленнях. Для взаємодії модулів використовується програмна шина (software bus). Є можливість вибрати і вбудувати модулі в систему під час компіляції. Такими модулями можуть бути файлова система (pHILE +), відладчик (pROBE +), мережеві протоколи (pNA +), бібліотека віддалених викликів процедур (pRPC +) і стандартна бібліотека ANSI C (pREPC +). Ці компоненти показані на рис.10.

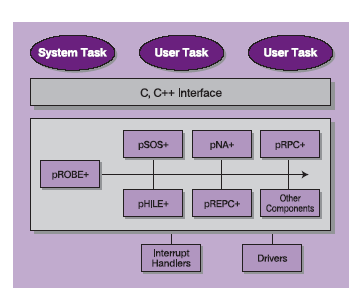


Рис.9. Компоненти pSOSsystem.

Виклики різних додатків здійснюються через програмні переривання.

pSOS + m є багатопроцесорної версією ядра pSOS +. Вона вимагає, щоб один вузол був головним, а інші - підлеглими. До цього ядра додані системні виклики, що дозволяють оперувати через кордони процесора.

У pSOS + не використовується поняття процесу, замість цього вона оперує завданнями, що відповідає поняттю потоків, що виконуються в одному процесі. Всі системні об'єкти розділяються між всіма потоками. Так як всі потоки розділяють один і той же контекст, час перемикання потоків стає дуже малою.

pSOSsystem має несегментоване модель пам'яті. Захист пам'яті може бути забезпечена через бібліотеку управління пам'яттю. Код, дані і стеки можна захистити за допомогою визначення відображень захисту пам'яті для кожного завдання. При цьому відповідальність лягає на розробника додатків, а це є непростим завданням. pSOSsystem пропонує дві абстракції для управління пам'яттю - регіони і розділи. Регіони - це шматки пам'яті нефіксованого розміру, в той час як розділи - шматки фіксованого розміру. Управління пам'яттю з допомогою розділів забезпечує швидке виділення пам'яті.

Управління переривань в pSOSsystem досить примітивне. Крім того, відсутні мьютекс і механізм успадкування пріоритетів, що може призвести до інверсії пріоритетів.

## 11. INTEGRITY

Продукт INTEGRITY (компанія Green Hills Software) [INTEGRITY] - це ОСРВ з передбачуваним часом відгуку, розрахована на застосування в тих ситуаціях, коли необхідні масштабованість ОС, її компактність і можливість роботи в режимі реального часу. Платформа INTEGRITY побудована на базі мікроядра velOSity [Velosity] і добре підходить для використання в недорогих пристроях з обмеженими апаратними ресурсами (сюди належить велика частина споживчої електроніки). Для своєї операційної системи компанія Green Hills пропонує інтегроване середовище розробки MULTI, повністю автоматизує процес створення ПЗ. Підтримуючи багатомовну розробку і налагодження, графічний інтерфейс пакета MULTI дає користувачу швидкий і зручний доступ до оптимізацією C / C + + компіляторами і функцій MISRA C. У цьому інструментальному пакеті міститься відладник рівня вхідної мови, компонувальник, аналізатор подій, Профілювальники продуктивності, програма виявлення помилок періоду виконання і засіб налагодження, не порушує основний режим функціонування.

Об'єктно-орієнтований підхід до проектування INTEGRITY забезпечує суворий контроль доступу та верифікацію безпеки і цілісності даних, взаємодій, компонентів і системи в цілому.

INTEGRITY використовує апаратну захист пам'яті і забезпечує підтримку численних захищених віртуальних адресних просторів, кожне з яких може містити кілька завдань програми. Ядро INTEGRITY оперує в своєму власному захищеному адресному просторі.

Для управління пам'яттю INTEGRITY використовує механізм віртуальної пам'яті. Щоб гарантувати абсолютну мінімальний час обробки переривань, ядро ніколи не блокує переривання, навіть при обробці критичних структур даних.

Ядро також уникає довгих обробок переривань. Як приклад таких переривань згадуються операції ділення і обробки рядків.

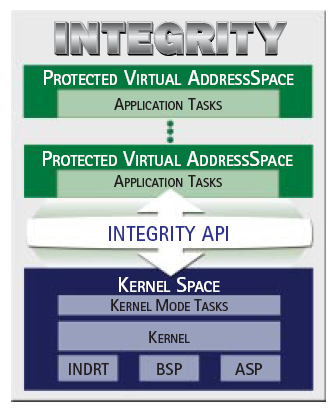


Рис.10. Структура INTEGRITY.

ОСРВ INTEGRITY включає дворівневий планувальник ARINC-653, заснований на сегментації (Partition Scheduler), який забезпечує гарантоване тимчасове вікно центрального процесора для кожної виконується завдання. Наприклад, якщо виконуються дві задачі, A і B, і кожній надано по 50% часу, то породження завданням B завдань B1 і B2 не вплине на виконання завдання A, оскільки час центрального процесора, виділене для задачі В (50%), розділиться на 3 для завдань В, B1 і B2, а для задачі A залишаться її колишні 50%. Таким чином, дії однієї задачі ніколи не зможуть вплинути на виконання інших завдань, що дозволяє уникати дії зловмисного коду, вірусів, проникнення хакера або просто помилок в інших адресних просторах.

## 12. LynxOS

Операційна система LynxOS RTOS (LynuxWorks, Inc) Є операційною системою жорсткого реального часу, яка призначена для спеціалізованої і телекомунікаційної апаратури [LynxOS]. Ця ОС є повністю детермінованою і володіє POSIX-, UNIX-і Linux-сумісністю. Областями застосування ОС LynxOS є також складні системи безпеки.

Остання випущена версія цього бренду ОС LynxOS-178 2.0 характеризується виробником як комерційна операційна система, що забезпечує високий рівень надійності та оперативності, необхідну для вбудованих додатків з особливими вимогами до безпеки. У LynxOS-178 2.0 реалізована підтримка інтерфейсу APEX (APlication / EXecutive - інтерфейс програми / керуючої програми) специфікації ARINC-653. Це означає, що дана операційна система відповідає найсуворішим вимогам до безпеки і надійності електронних систем для військової та цивільної авіації. Система LynxOS-178 2.0 повністю відповідає положенням рівня А специфікації DO-178B.

ОСРВ LynxOS-178 2.0 відповідає вимогам стандартів POSIX і ARINC-653, а також DO-178B, що означає гарантію переносимості прикладного коду вбудованих систем, багаторазового використання створених програм, а також відповідність найсуворішим нормативам операційних систем з підвищеними вимогами до безпеки. Використання LynxOS-178 2.0 дозволяє застосовувати будь-які раніше сертифіковані програми і розробки.

## 13. Microware OS-9

Операційна система реального часу OS-9 корпорації Microware System є багатозадачного, розрахованої на багато користувачів операційною системою для вбудованих додатків, що працюють в режимі реального часу [OS-9]. Ця система призначена для роботи в таких системах, як мобільні телекомунікаційні пристрої, що вбудовуються термінали доступу в Інтернет, інтерактивні цифрові телевізійні приставки. OS-9 працює на таких процесорах, як Motorola 68K, ARM / StrongARM, Intel IXP1200 Network Processor, MIPS, PowerPC, Hitachi SuperH, x86 or Intel Pentium, Intel IXC1100 XScale.

Ядро OS-9 є маштабованим, повністю витісняється, підтримує функціонування до 65535 процесів, надає 65535 рівнів пріоритету і забезпечує роботу до 255 користувачів. Ядро OS-9 містить більше 90 системних викликів, які дають можливість керувати динамічним режимом диспетчеризації, розподілом пам'яті, міжпроцесорного комунікацією і т.д. - Аж до управління вбудовуваним в ядро ОС режимом економічного споживання харчування. Характеристики продуктивності ядра: 5,6 мкс - час затримки переривання (Interrupt Latence Time), 14 мкс - час перемикання контексту процесу (для процесора MC68040, 30MHz).

Система введення-виведення ОС підтримує такі формати пристроїв масової пам'яті та основних інтерфейсів периферійних пристроїв: Raw, MS-DOS, True FFS, CardSoft PCMCIA, USB, IrDA.

Середа OS-9 підтримує кілька програмних комунікаційних платформ - mwSoftStax (Microware), Harris & Jeffries, Trillium. Завдяки наявності стандартизованої комунікаційного середовища в OS-9 працюють сучасні і найбільш перспективні комунікаційні протоколи: ISDN, ATM, X.25, MPEG-2, FR, SS7 і т.д.

Графічні засоби в OS-9 представлені різноманітними продуктами - від компактних мінімізованих по ресурсах програмних модулів підтримки графіки Multimedia Applications User Interface (MAUI) фірми Microware до повнофункціональних клієнт-серверних систем графічних G-Windows (GESPAC), XiBase9 GUI (XiSys), MGR (Reccoware).

Корпорація Microware однією з перших ліцензувала Java для вбудованих додатків і є лідером за пропозицією різноманітних засобів та програм в рамках OS-9 для різних класів пристроїв. У OS-9 користувачеві пропонується Java VM, Java-Compiler/JIT, Java-ROMizer, Java Applets Lib, Embedded Java, Personal Java.

У різних областях застосування для портування OS-9 на апаратну платформу виробника використовуються наступні програмні пакети:

OS-9 for Embedded Systems Kit,

OS-9 for Communications Systems,

OS-9 for Consumer Devices (Wireless Devices),

OS-9 for Interactive Digital TV,

OS-9 Java Starter Kit.

В якості інтегрованої крос-середовища розробки додатків для OS-9 корпорація Microware розробила середу Hawk, яка функціонує на платформі MS Windows NT. Hawk є відкритою середовищем і надає стороннім розробникам інструментальних засобів більше сотні API, що дозволяють включати до складу середовища Hawk продукти відомих фірм розробників інструментального ПЗ.

Для потреб спільної програмно-апаратної розробки в Hawk вбудовані засоби для роботи з внутрішньосхемними емуляторами серії visionICE фірми EST. Є кошти налагодження в режимі реального часу.

Для тестування та верифікації ПЗ розроблено засіб верифікації програмного забезпечення CodeTEST (Applied Microsystems), що вбудовуються в Hawk. Це засіб дає можливість здійснювати трасування вбудованого ПЗ та контролювати його характеристики, а також хід виконання тестів і розподіл пам'яті.

## 14. GRACE-OS

Система GRACE-OS являє собою планувальник CPU в режимі м'якого реального часу для мобільних пристроїв, що виконують, головним чином, мультимедійні програми [YN03]. Система GRACE-OS розроблена в Іллінойському університеті (University of Illinois, Department of Computer Science). При проектуванні системи першочерговими цілями ставилися завдання підтримки якості сервісу і заощадження енергії. Для досягнення поставлених цілей GRACE-OS інтегрує динамічне маштабування напруги в диспетчеризацію на основі моделі м'якого реального часу і визначає, як швидко, коли і як довго має здійснюватися виконання додатків. Планувальник GRACE-OS реалізований всередині ядра Linux, і апробовано на лептопі HP Pavilion.

Планувальник GRACE-OS складається з трьох основних компонентів - профайлера, планувальника SRT (soft real-time) і адаптера швидкості, як показано на рис.11.

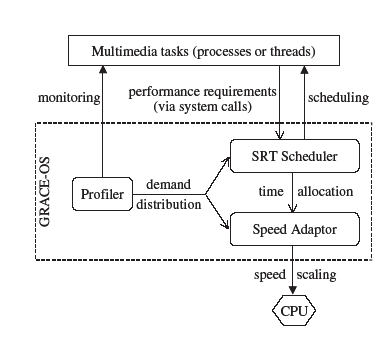


Рис.11. Архітектура GRACE-OS

Вдосконалений планувальник виконує планування в режимі м'якого реального часу і динамічне масштабування напруги.

Профайлера здійснює моніторинг коефіцієнта завантаження циклу окремих завдань і автоматично отримує розподіл вірогідності їх запитів всередині циклу в залежності від коефіцієнта завантаження. Планувальник SRT відповідає за виділення циклів завданням і їх планування, забезпечуючи необхідну продуктивність. Планування в режимі м'якого реального часу засноване на статистичних вимогах продуктивності і розподілі запитів кожного завдання. Адаптер швидкості динамічно регулює швидкість CPU, забезпечуючи економію енергії. Він адаптує швидкість виконання кожного завдання на основі розподілу виділяється завданням часу, що забезпечується планувальником SRT, і розподілу запитів, що забезпечується профайлера.

## 15. C EXECUTIVE

C EXECUTIVE (JMI Software Systems, INC) [CEXEC] - це многозадачное ядро реального часу для вбудованих систем, що працює на 8 - , 16 - і 32-бітових CISC процесорах, на широкому діапазоні RISC процесорів і DSP (Digital Signal Processor). Це ядро забезпечує швидке перемикання контексту, має маленький розмір. Над ядром можна надбудовувати DOS-сумісну файлову систему, TCP / IP і SNMP.

Ядро C EXECUTIVE володіє високим ступенем маштабованості, можна навіть сказати, що маштабованість внутрішньо властива такого ядра, оскільки набір системних викликів компонується з бібліотеки під час створення системи, і виконується екземпляр системи буде містити тільки ті системні виклики, які використовуються конкретним додатком. До того ж таке ядро можна конфігурувати з включенням або без включення квантування часу, генератора тактових імпульсів, сигналів і т.п., дозволяючи, таким чином, здійснювати вкрай високу оптимізацію системної конфігурації для невеликих цільових систем.

Ядра реального часу компанії JMI застосовуються в сотнях вбудованих додатків, включаючи лазерні принтери, електронні касові апарати, медичну апаратуру, пристрої комунікації, військові та космічні програми та інші критичні за часом системи.

## 16. CMX-RTX

Операційна система CMX-RTX [CMXRTX] є багатозадачного операційною системою реального часу для мікроконтролерів, мікропроцесорів, мікрокомп'ютерів і DSP (Digital Signal Processor). Ця система підтримує вкладені переривання, має малий час перемикання контекстів, низькі часи затримок переривань і вкрай малі розміри. Планувальник завдань і компонент управління переривань написані на мові асемблера для прискорення обчислювального процесу. CMX-RTX має компоненти управління завданнями, подіями, часом, повідомленнями, чергами, ресурсами, семафорами, фіксованими блоками пам'яті, автоматичним вимиканням живлення, асинхронної послідовної передачею даних (UART - universal asynchronous receiver-transmitter), пріоритетними переривань.

## 16.1 CMX-TINY +

CMX-TINY + [CMXTINY] є багатозадачного операційною системою реального часу для широкого ряду мікропроцесорів і мікрокомп'ютерів, яка створена для розробки додатків, що виконуються під ОСРВ і використовують тільки вбудовується пам'ять процесора. Ця система забезпечує незначно меншу функціональність, ніж система CMX-RTX. Вона створювалася для того, щоб її можна було помістити всередині невеликий бортовий пам'яті RAM (random access memory) в чіпі, яка має розмір 512 байтів і більше.

## 17. Inferno

Inferno (корпорація Lucent) - це компактна операційна система, створена для побудови розподілених мережевих систем на широкому діапазоні пристроїв та платформ [INFERNO]. Ця система має міжплатформову переносимістю і може виконуватися як для користувача додаток або як незалежна операційна система. Підтримується для більшості широко поширених операційних систем і платформ. Кожна система Inferno надає користувачеві ідентичну середовище розробки незалежно від основної операційної системи чи архітектури, дозволяючи працювати у гомогенній середовищі з безліччю різних платформ.

Inferno - це не просто операційна система; вона також є повноцінною середовищем розробки, забезпечуючи всі кошти, необхідні для створення, налагодження та тестування додатків. Програми, що створюються в середовищі Inferno, пишуться на мові Limbo, який є модульним паралельним мовою програмування з C-подібним синтаксисом. Код на Limbo компілюється в архітектурно-незалежний байт код, який потім може бути виконаний у режимі інтерпретації (або код компілюється оперативно) для цільового процесора. Таким чином, Inferno-додатки виконуються ідентично на всіх Inferno-платформах.

Inferno пропонує повну прозорість ресурсів і даних, застосовуючи якусь систему іменного простору. Ресурси представляються як файли, застосовується один стандартний комунікаційний протокол. Завдяки цьому такі ресурси, як сховища даних, сервіси та зовнішні пристрої, можуть розділятися між різними Inferno-системами. Інтерфейс ресурсу можна імпортувати в локальну систему, і їм можуть користуватися додатки, які не знають, чи є даний ресурс локальним або віддаленим.

Безпека високого рівня також є частиною Inferno-системи. Завдяки тому, що для всієї мережі використовується один стандартний комунікаційний протокол, безпека забезпечується на системному рівні. Inferno пропонує також підтримку аутентифікації, заснованої на шифруванні.