1 手持设备的基本概念和应用

- 基本概念 – 理解

  A *mobile device* (also known as *cellphone device*, *handheld device*, *handheld computer*, *"Palmtop"* or simply *handheld*) is a pocket-sized computing device, typically having a display screen with touch input or a miniature keyboard.

  手持设备，又称为 手机设备，掌上设备，掌上电脑，是一种小型的计算设备，典型的有一个显示屏，支持键盘输入或触摸式输入，随首技术的革新，现在的手持设备大都提供虚拟键盘。

- 应用 – 举例

  In the case of the *personal digital assistant (PDA)* the input and output are combined into a touch-screen interface.

  - *Smartphones* and *PDAs* are popular amongst those who require the assistance and convenience of a conventional computer, in environments where carrying one would not be practical.

  Enterprise digital assistants can further extend the available functionality for the business user by offering integrated data capture devices like Bar Code, RFID and Smart Card readers.

- 相关知识和技术
  - 4 种嵌入式软件体系结构
  - RTOS or RTK

嵌入式软件体系结构

➢ 轮转结构:轮转结构(round-robin)最简单的一种结构

主循环依次检查每个 I/O 设备，并为需要服务的设备提供服务。 不存在中断

优点 结构简单

缺点 最坏响应时间 缺乏优先级 结构缺乏可扩展性

➢ 带中断的轮转结构:

中断程序处理硬件特别紧急的需求，然后设置标记；

主循环轮询这些标记，然后根据这些需求进行后续的处理。

优点:为硬件操作提供优先级。实质上，中断程序中的所有操作拥有比主程序中任务代码更高的优先级。

缺点 :结构带来一定的复杂度 共享数据问题

优先级机制不彻底！所有任务代码以同样的优先级来执行。 任务的最坏响应时间：

➢ 函数队列调度结构

更复杂精细的结构

中断程序在一个函数指针队列中添加一个函数指针，以供主程序调用。

主程序仅需要从该队列中读取相应的指针并且调用相关函数。

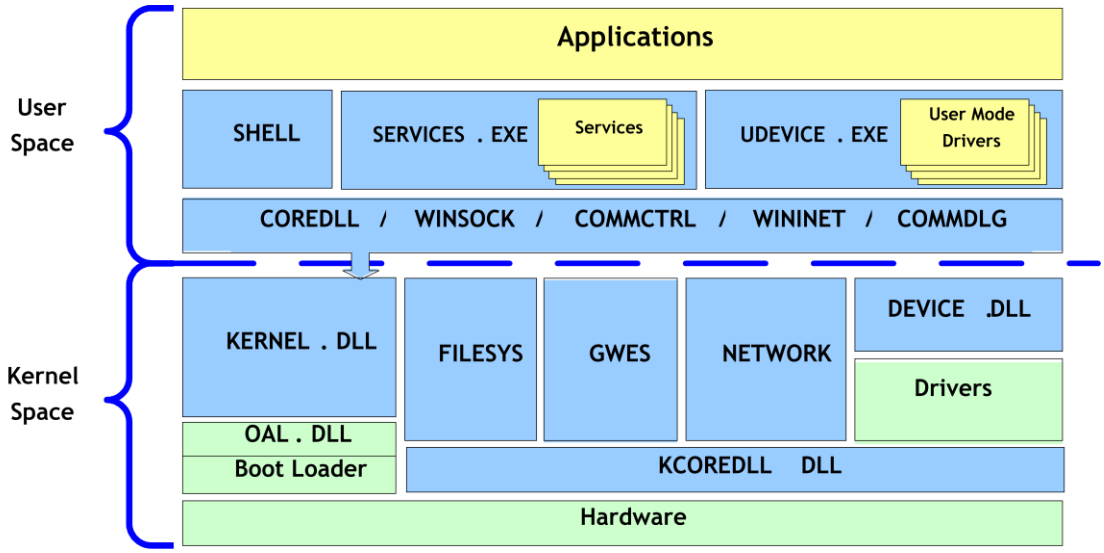
引进任务优先级：通过对函数指针排队实现
优点 :提供了任务优先级
缺点:程序复杂　　最坏响应时间？

- 实时操作系统 RTOS
  明确提出“任务”概念
  中断程序和任务代码之间的必要信号发送是通过实时操作系统处理的，并不需要使用共享变量来达到这个目标。
  代码中并没有用循环来决定下一步要做什么。实时操作系统内部的代码决定什么任务代码可以运行。
  操作系统可以在一个任务运行期间将其挂起，以便运行另一个任务。
  结构可扩展性强。

结构种类	是否允许优先级	任务代码的最坏响应时间	代码改变时响应时间的稳定性	简单性
轮转结构	不允许	所有任务代码的总和	差	很简单
带中断的轮转结构	中断程序有优先级次序，所有任务代码在同一个优先级上	所有任务代码的执行时间的总和(加上中断程序的执行时间)	中断程序响应时间的稳定性好；任务代码响应时间的稳定性差	必须处理中断程序和任务代码的共享数据
函数队列调度结构	中断程序有优先级次序，任务代码也有优先级次序	最长函数的执行时间(加上中断程序的执行时间)	相对较好	必须处理共享数据，并且要编写函数排队代码
实时操作系统结构	中断程序有优先级次序，任务代码也有优先级次序	0(加上中断程序的执行时间)(对优先级最高的任务来说)	很好	最复杂（尽管多数复杂部分是在操作系统内部）

2 Windows CE 体系结构

（The CE system architecture is shown in Figure 6.2. Items in blue are provided with the OS, the OEM typically provides the items shown in green, and applications programs provide the items shown in yellow.）了解一下



- 体系结构的特点
  - 分层结构（使系统有良好的可扩展性和可维护性）
    - Hardware, OEM, OS layer, Application Layer
  - 微内核特点：内核只实现那些必须由内核实现的基本功能，而将图形系统文件系统，设备驱动及通信等功能放在内核之外，以系统服务的形式提供各种功能。（这种结构的优点是有一个精炼的内核，便于裁剪与移植，而且系统服务运行在用户地址空间，因个别驱动程序的错误不至于导致整个系统崩溃，其不足之处是在运行中用户状态与内核态须频繁切换，从而导致系统效率不如单体内核。）

CE6 differences：（到网上找找=。=）：

（“CE” means - Compact Edition?Consumer Edition?）

1． the old "kernel mode" that you knew from CE5 does not exist in CE6. SetKMode and ALLKMODE do not exist. User mode in CE6 refers to running in any other process.

2．In CE5 the kernel and OEM Adaptation Layer (OAL) linked together to make nk.exe. In CE6 we separated the OAL and kernel into separate binaries, oal.exe (which ends up getting renamed to nk.exe) and kernel.dll. The Kernel Independent Transport Layer (KITL) was also separated into its own library, kitl.dll. (好处：This change was made primarily for improved updateability. In the past, if Microsoft released a kernel update, the OEM would have to take this and link it again with their OAL to produce a new nk.exe. Now the OEM only has to distribute Microsoft’s new kernel.dll. The other benefit of this change is that it formalizes the interface between the kernel, OAL and KITL.)

3With CE6, many critical OS components moved into the kernel process.

filesys.exe --> filesys.dll

device.exe --> device.dll

gwes.exe --> gwes.dll

Additionally, all of the drivers that used to load into those processes now load into the kernel process by default.

3 进程、线程和调度

- 进程和线程的基本概念

  进程：是程序一次动态执行实例，是动态的，有创建执行结束完整的生命周期。包含下面两个东东：

  - A kernel object that the operating system uses to manage the process. The kernel object is also where the system keeps statistical information about the process.
  - An address space that contains all the executable or DLL module's code and data. It also contains dynamic memory allocations such as thread stacks and heap allocations.

  线程：是 windows CE 最小的可执行单元。

- 进程与内存管理

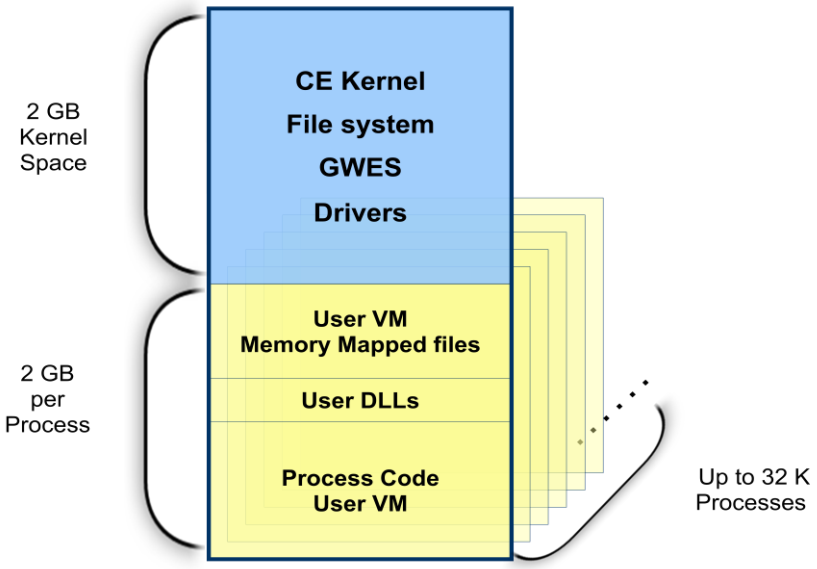


Figure The Windows Embedded CE 6.0 Virtual Memory Space Model.

2GB user space, 2GB kernel space

CE 5.0：32 processes; 32MB each process.

CE 6.0：32000 processes, 2GB each process.

- CE6.0: 2GB of Virtual Memory per process;Up to 32,000 Processes

  Each Process has one or more threads;Thread – basic unit of execution managed by scheduler;Many threads per process – limited by system resources (memory)

  CE5.0:最多支持 32 个进程可同时运行，每个进程占据 32MB 的虚拟地址空间，称为一个 slot。系统启动时 filesys.exe,gwes.exe,device.exe 已中据多个 slot ,因此，用户实际可用的进程数不到 30 个，所以在构建系统时应尽量选择多线程解决方案而不是采用多进程。

- 线程调度（听说优先级反转不考）

  Windows Embedded CE 6.0 uses a priority-based time-slice algorithm to schedule the execution of threads.

  - Preemptive scheduling with 256 priority levels
  - Round-robin scheduling of threads with same priority
  - Highest Priority 0 threads run to completion
  - Time slice 1-100ms (100 ms is Default)

- 线程同步的机制:

  Windows CE 提供了 Mutex,Event,Semaphore 三种内核机制来实现线程中的同步，所有同步对象都 有两种状态，signaled or Non-signaled.未通知状态表示该同步对象被其他的线程占用，不能被其他的等待线程占用。当其变为通知状态时，等待在它上面的阻塞线程会得到通知，并且转为就绪态，等待调度执行。这种方法可以使线程锁步执行，这就是同步的原理。

- Critical Sections

临界区对象运行在用户模式。它能保证在临界区内所有被访问的资源不被其它线程访问，直到当前线程执行完临界区代码。除了 API 外，MFC 也对临界区函数进行了封装
使用临界区要注意的是避免死锁。当有两个线程，每个线程都有临界区，而且临界区保护的资源有相同的时候，这时就要在编写代码时多加考虑。

- Mutexes

- Semaphores (counting)

- Events – used to indicate something happened

事件(Event)是 Win32 提供的最灵活的线程间同步方式。运行在内核模式。

事件的两种状态：激发状态(signaled or true)　未激发状态(unsignal or false)

两类事件

手动设置：这种对象只能用程序来手动设置，在需要该事件或者事件发生时，采用 SetEvent 及 ResetEvent 来进行设置。

自动恢复：一旦事件发生并被处理后，自动恢复到没有事件状态，不需要再次设置。

(同步机制:事件对象、信号量对象、互斥量对象、临界区、互锁函数(原子操作)、消息队列 线程同步的一些解决方法运行在用户模式，另外一些运行在内核模式？哪些不同？)---google 一下？

- 用户方式中的线程同步
  - 允许线程保留在用户方式下实现同步。如互锁函数，CRITICAL_SECTION。
  - 优点：速度快
  - 局限：互锁函数只能在单值上运行；CRITICAL_SECTION 只能对单个进程中的线程同步。
- 使用内核对象进行同步
  - 优点：适应性广泛
  - 缺点：速度慢

- 可用于同步的内核对象：
  - Processes, Threads, Jobs, Files;Events; Semaphores, Mutexes ;File change notifications;Waitable timers ;Console input

- 进程间通信

- Memory mapped files and Pointers ：文件映射

通过内存映射文件可在进程的共享虚拟地址空间内保留一个地址空间的区域，同时

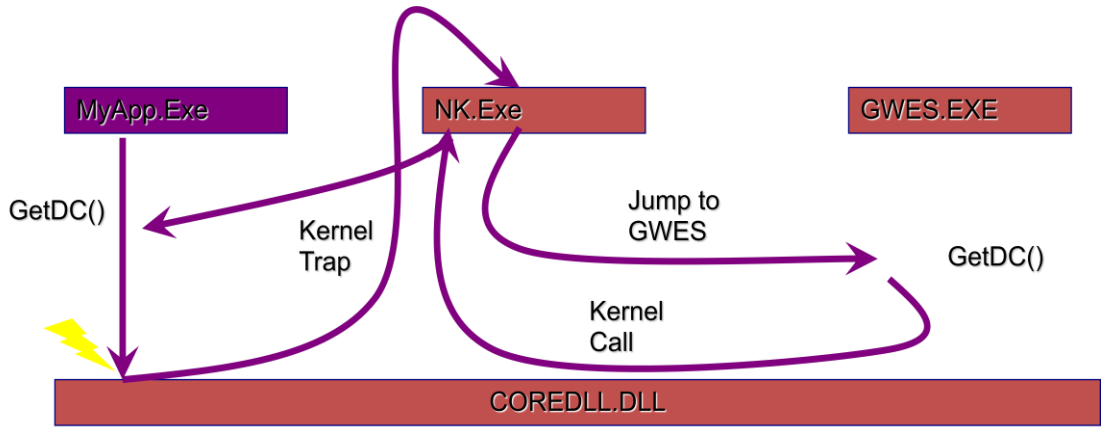
将文件所在的物理内存映射到此区域，通过对虚拟内存进行读写操作就能实现对文

件的操作，而且文件何时被加载到内存，何时被换出完全不需要开发人员关心，由 os 内存管理来实现。

- Point to Point Message Queue APIs：点对点消息队列

  通常队列是一个先进先出的队列结构，当一个进程把消息写入队列，需要此消息的其他的进程，就可以从队列中取出消息，从而达到进程间通信的目的。

System API call 的流程



1 process calls a system call

2 calls into a wrapper function

3 That function prepares the function's parameters for the kernel and causes a software exception to occur

4 The kernel then handles this exception and determines the correct destination process to send the function call request to or which .exe file can fulfill the request.

5 The process that owns the function executes it using the same stack and register values that the original thread contains in the calling process

（ **Interrupt Processing**：了解一下，没有说考，只是作业上提到了

- Interrupt Service Handler (ISH) in Kernel decides which ISR to call

- Uses an Interrupt Service Routine (ISR) and an Interrupt Service Thread (IST)
- ISR handles basic interrupt tasks quickly, starts IST and returns
- IST can complete remainder of task later
- Using both an ISR and IST reduces Interrupt Latency)

5

存储管理和文件系统

- 由Filesys.exe管理（->Filesys.dll in CE6）
- 对象存储的概念
  - （物理上）对象存储是一个被Filesys.exe管理的内存堆；如果有后备电源，它可用于为应用程序提供永久存储。
  - （逻辑上）可分为：RAM文件系统、注册表和CE数据库。
  - 对象存储中的数据存储是基于事务的。
- 操作系统使用对象存储技术完成以下任务：
  - 管理栈和内存堆；按需求压缩或展开文件；无缝地集成基于ROM的数据和基于RAM的数据。

对象存储与文件系统（续）

- 文件系统
  - Built-in 文件系统
    - ROM-RAM文件系统（\目录）
    - ROM-only文件系统（\Windows目录）
  - 可安装的文件系统（\Storage Card, \CD Driver, etc.）
- 注册表
  - 基于RAM的注册表
  - 基于Hive的注册表
- CE数据库(略)

XIP

- XIP：Execute In Place（本地执行）
  - 允许程序代码不进入RAM，直接在ROM/Flash中执行
- 优点：
  - 代码段不必先加载到物理内存中，节省内存。（在Windows CE中，OS分配虚拟地址空间给代码段，并将其映射到ROM/Flash中）
- 缺点：
  - 只支持允许线性访问的ROM/Flash（e.g. NOR Flash）
  - 执行速度相对较慢；不适合实时性要求高的场合。

WinCE 启动流程

1. CPU 加电，跳转到复位向量
2. [可选] 引导程序从 Startup()开始执行
3. 执行 OAL 中的 Startup()
4. KernelStart() [KernelInitialize() For x86]
5. Kernel 调用 OAL 中的 OEMInit()
6. 完成内核初始化
7. 内核加载 Filesys.exe
8. FileSys 初始化注册表
9. 内核加载在 HKEY_LOCAL_MACHINE\Init 中列出的应用程序

WinCE build可能产生的文件

文件名	描述
nk.bin	必需。按照section组织的二进制操作系统映像
nk.nb0	可选。可直接烧到flash中，支持XIP的操作系统映像
eboot.bin	可选。按照section组织的二进制以太网Bootloader映像
eboot.nb0	可选。可直接烧到flash中，支持XIP的以太网Bootloader映像

5 GWES: 结合 Windows 窗口应用程序:

图形系统依赖窗口的设备的上下文来绘图 ,而窗口也需要图形系统来绘制自身 ,在 windows 中事件是通过消息机制实现的。消息总是发送给某个窗口，而每个窗口都有一个 WinProc 来处理消息。

设备管理器和服务管理器 (此部分不清楚答案是什么)

- 设备管理器：device.exe（device.dll in CE6）
  - 作用和流程

    负责加载和管理 windows ce 下绝大多数的设备驱动程序,包括网卡驱动，电池驱动，声卡驱动，串口驱动，usb 设备驱动及 pcmcia 驱动

    流程： 内核-> device.exe->i/o 资源管理器 Iorm.lib->电源管理器 Pm.dll-> 总线枚举器 BusEnum.dll –> PCIBUS.dll
  - 驱动程序的编写：动态链接库
  - 相应的注册表项
- 服务管理器：services.exe
  - 作用和流程:

    Services.exe 负责加载所有的服务

    服务的启动有两种方式，通过注册表中设置相应的注册表项，系统启动时让 services.exe 加载服务;另外一种是应用程序使用函数手动加载服务。
  - 服务程序的编写：动态链接库
  - 相应的注册表项(Context 项目,可配置为普通服务,独立服务,超级服务）

    如果 某个服务 dll 在单独的进程中被加载，那么此服务被称为独立服务。

    超级服务侦听某个网络端口，当有客户连接这些端口时，服务程序会给客户提供相应的服务。

    （只需把 HKEY_LOCAL_MACHINE\Services\<servernname>\下的 Context 项设置为 1）

6 应用程序的开发

（Why Platform Manager?

WinCE 应用程序开发与 Windows 桌面应用程序开发的主要不同：

- 在开发桌面应用程序时，应用程序同时在桌面操作系统中运行;
- 在开发 WinCE 应用程序时，需要将 WinCE 应用程序 download 到 WinCE 目标设备上运行，并且调试的情况也是一样。）

```
#include <windows.h>
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR szCmdLine,
int nShow)
{
  MessageBox(NULL, TEXT("Hello, Win32"), TEXT("HelloMsg"), MB_OK);
  return 0;
}
```

- WinMain() prototype

  int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR szCmdLine, int nShow)

Why?

  - Under Windows CE, the command-line string is a Unicode string. In all other versions of Windows, the string is always ASII.

7 系统定制与开发：

- 系统定制与开发的整体流程：

  系统定制：1 得到并安装 bsp ;2 定制操作系统; 3 下载到开发板上运行调试：3.1 得到并安装 bootLoader 3.2 配置网络连接 3.3 配置调试串口 3.4 配置 platform builder 3.5 连接设置 3.6 下载运行映像 ； 4 发布操作系统

  开发应用程序大致可以分三个步骤：

  1 安装合适的 sdk 2 编写代码和调试 3 发布应用程序

- Build System – 4 steps
  - 与 Build 有关的文件：Source，DIRS，Makefile
  - 与系统初始化有关的文件：.bib, .reg, .dat, .db

  1Sysgen 阶段（生成阶段）2 Feature build 阶段 （编译阶段）

  3 Release copy 阶段（release 文件夹复制阶段） 4 Make Image 阶段（镜像打包阶段）

8 BSP

- BSP 的概念与组成：4 部分

  BSP：Board Support Package

  - 主板硬件和操作系统之间的一层软件系统。严格地说，BSP 属于操作系统的一部分
  - 解决操作系统跨不同 CPU 体系结构的方法之一：抽象操作系统和硬件之间的交互接口

板级支持包的组成

内容	描述
引导程序	**加载操作系统映象**
OEM 抽象层 (OAL)	**连接内核映象，支持硬件的初始化和管理**
设备驱动	**支持相关外围设备以及动态安装的设备**
配置文件	**可以通过对环境变量、.bib文件和 and .reg文件的修改来重新配置BSP**

- Bootloader 的简单分析:功能：初始化目标硬件设备，控制启动过程，下载并执行操作系统映像
  - 代码框架：blcommon　oem 代码 eboot 存储管理　EDBG 驱动程序
  - 程序流程

Boot Loader Startup Sequence			
Startup()			
	EbootMain()		
		BootloaderMain()	
		OEMDebugInit()	
		OEMPlatformInit()	
		OEMPreDownload()	
			Download occurs
		OEMLaunch()	

- 黑体字标识的函数需要由 OEM 厂商来实现.

- 用户需做的：

  1 实现 OEM 的应用程序接口（API）.

  2 连接 Microsoft 提供的库

引导程序 – StartUp 函数

- 硬件复位和运行时复位需要执行的第一条指令
- 设置为超级用户模式
- 执行必须的硬件初始化:
  - CPU 内存控制器 系统时钟 串口 缓存 快表 (TLBs)
- 根据使用的 CPU 修改 Startup.s

引导程序 – EbootMain

- EbootMain 是 C 代码运行的入口
- 调用 BLCOMMON 库
- BLCOMMON 库 源文件在 Blcommon.c 文件中，路径为%_WINCEROOT%\Public\Common\Oak\Drivers\Ethdbg directory

引导程序 – OEMDebugInit

- 用来初始化串行口，作为调试输出
- OEMDebugInit 初始化完成后, 一个 Windows CE 的标记会出现，表示这个接口可以使用了.

引导程序 – OEMPlatformInit

- 各种 OEM 硬件平台初始化函数，包括时钟, PCI 接口,或者 NIC 接口.
- NIC 接口用于下载映象，另外服务于后面一些函数.

引导程序 – OEMPreDownload

- 在加载一个运行时映象时首先被 BLCOMMON 调用.
- 查找硬件设备的 IP 地址，并与宿主机相连
- 如果出错返回-1

引导程序 – OEMLaunch

- OEMLaunch 是引导程序的最后一个需要运行的函数.
- 负责跳转的到需要运行的映象.
- 跳转到由 dwLaunchAddr 指定的第一条指令，这条指令在运行时映象的启动函数里.