Linux嵌入式系统的内存管理方法详细介绍 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/253/2021_2022_Linux_E5_B5_8C_E5_85_c103_253072.htm 1 嵌入式系统中对内存分配的要求 ①快速性。嵌入式系统中对实时性的保证，要求内存分配过程要尽可能地快。因此在嵌入式系统中，不可能采用通用操作系统中复杂而完善的内存分配策略，一般都采用简单、快速的内存分配方案。当然，对实性要求的程序不同，分配方案也有所不同。例如，VxWorks采用简单的最先匹配如立即聚合方法；VRTX中采用多个固定尺寸的binning方案。 ②可靠性。也就是内存分配的请求必须得到满足，如果分配失败可能会带来灾难性的后果。嵌入式系统应用的环境千变万化，其中有一些是对可靠性要求极高的。比如，汽车的自动驾驶系统中，系统检测到即将撞车，如果因为内存分配失败而不能相应的操作，就会发生车毁人亡的事故，这是不能容忍的。 ③高效性。内存分配要尽可能地少浪费。不可能为了保证满足所有的内存分配请求而将内存配置得无限大。一方面，嵌入式系统对成本的要求使得内存在其中只是一种很有限的资源；另一方面，即使不考虑成本的因素，系统有限的空间和有限的板面积决定了可配置的内存容量是很限的。 2 静态分配与动态分配 究竟应用使用静态分配还是动态分配，一直是嵌入式系统设计中一个争论不休的总是。 当然，最合适的答案是对于不同的系统采用不同的方案。如果是系统对于实时性和可靠性的要求极高（硬实时系统），不能容忍一点延时或者一次分配失败，当然需要采用静态分配方案，也就是在程序编译时所需要的内存都已经分配好了。例如，火

星探测器上面的嵌入式系统就必须采用静态分配的方案。另外，WindRiver公司的一款专门用于汽车电子和工业自动化领域的实时操作系统OSEKWorks中就不支持内存的动态分配。在这样的应用场合，成本不支持内存的动态分配。在这样的应用场合，成本不是优先考虑的对象，实时性和可靠性才是必须保证的。当然，采用静态分配一个不可避免的总是就是系统失去了灵活性，必须在设计阶段就预先知道所需要的内存并对之作出分配；必须在设计阶段就预先考虑到所有可能的情况，因为一旦出现没有考虑到的情况，系统就无法处理。这样的分配方案必须导致很大的浪费，因为内存分配必须按照最坏情况进行最大的配置，而实际上在运行中可能使用的只是其中的一小部分；而且在硬件平台不变的情况下，不可能灵活地为系统添加功能，从而使得系统的升级变得困难。 大多数的系统是硬实时系统和软实时系统的综合。也就是说，系统中的一部分任务有严格的时限要求，而另一部分只是要求完成得越快越好。按照RMS（Rate Monotoin Scheduling）理论，这样的系统必须采用抢先式任务调度；而在这样的系统中，就可以采用动态内存分配来满足那一部分可靠性和实时性要求不那么高的任务。采用动态内存分配的好处就是给设计者很大的灵活性，可以方便地将原来运行于非嵌入式操作系统的程序移植到嵌入式系统中，比如，许多嵌入式系统中使用的网络协议栈。如果必须采用静态内存分配，移植这样的协议栈就会困难得多。另外，采用动态内存分配可以使设计者在不改变基本的硬件平台的情况下，比较灵活地调整系统的功能，在系统中各个功能之间作出权衡。例如，可以在支持的VLAN数和支持的路由条目数之间作出调整，或

者不同的版本支持不同的协议。说到底，动态内存分配给了嵌入式系统的程序设计者在比较少的限制和较大的自由。因此，大多数实时操作系统提供了动态内存分配接口，例如malloc和free函数。 3 RTOS提供的内存分配接口 不同的RTOS由于其不同的定位，采用不同的内存分配策略。 例如VRTX中，采用类似于GNU C中由Doug Lea开发的内存分配方案，即Binning算法，系统内存被分成了一些固定尺寸的内存块的算法，系统内存被分成了一些固定尺寸的内存块的集合。这种方法的优点是查找速度快而且不会产生内存碎片。但是，它的缺点也很明显，就是容易造成浪费，因为内存块的尺寸只有有限个，分配时只能取较大的内存块来满足一个较小的需求，累积起来，浪费就很大了；而且操作系统管理这样一个内存分配表也是一个很大的负担。 下面详细介绍一下我们常用的RTOS美国风河公司（WindRiver）的VxWorks中采用的内存分配策略。VxWorks的前身就是VRTX，据说VxWorks的名称来自make vrtx work。 VxWorks的内存管理函数存在于2个库中；memPartLib（紧凑的内存分区管理器）和memLib（完整的内存分区管理器）。前者（memPartLib）提供的工具用于从内存分区中分配内存块。该库包含两类程序，一类是通用工具创建和管理内存分区并从这些分区中分配和管理内存块；另一类是标准的malloc/free程序提供与内存分区的接口。系统内存分区（其ID为memSysPartId是一个全局变量）在内核初始化时由usrRoot调用memInit创建。其开始地址为RAM中紧接着VxWorks的BSS段之后，大小为所有空闲内存，如图1所示。 当创建其它分区时，一般需要先调用malloc从系统内存分区中分配一段内存才能创建。内存分区

的结构定义为mem_part，包含1个对象标记，1个双向链表管理空闲块，1个信号量保护该分区及一些统计信息，如总尺寸、最大块尺寸、调试选项、已分配的块数、已分配的尺寸等。其语句如下： typedef struct mem_part { OBJ_CORE objCore. /*对象标志*/ DL-LIST freeList； /*空闲链表*/ SEMAPHORE sem. /*保护分区的信号量*/ Unsigned totalWords； /*分区中字（WORD）数*/ Unsigned minBlockWords. /*以字为单位的最小块尺寸*/ Unsigned options. /*选项，用于调试或统计*/ /*分配统计*/ unsigned curBlocksAllocated. /*当前分配的块数*/ unsigned curWorkdAllocated. /*当前分配的字数*/ unsigned cumBlockAllocated； /*累积分配的块数*/ unsigned cumWordsAllocated； /*累积分配的字数*/ }PARTITION；一般系统中只有1个内存分区，即系统分区，所有任务所需要的内存直接调用malloc从其中分配。分配采用First-Fit算法（注意这种算法容易导致大量碎片），通过free释放的内存将被聚合以形成更大的空闲块。这就是VxWorks的内存分配机理。分配时可以要求一定的对齐格式。注意，不同的CPU架构有不同的对齐要求。为了优化性能，malloc返回的指针是经过对齐的，为此的开销随构不同而不同。例如，68K为4字节对齐，开销8字节；SPARC为8字节对齐，开销12字节；MIPS为16字节对齐，开销12字节；I960为16字节对齐，开销16字节。MemLib库中提供了增强的内存分区管理工具，并且增加了一些接口，而且可以设置调试选项。可以检测2类错误：①尝试分配太大的内存；②释放内存时发现坏块。有4种错误处理选项，当发生错误时记录消息或挂起任务。 但是，使用动态内存分配malloc/free时要注意到以下几方面的限制。①因为系统

内存分区是一种临界资源，由信号量保护，使用malloc会导致当前调用挂起，因此它不能用于中断服务程序；②因为进行内存分配需要执行查找算法，其执行时间与系统当前的内存使用情况相关，是不确定的，因此对于有规定时限的操作它是不适宜的；③由于采用简单的最先匹配算法，容易导致系统中存在大量的内存碎片，降低内存使用效率和系统性能。针对这种情况，一般在系统设计时采用静态分配与动态分配相结合的方法。也就是对于重要的应用，在系统初始化时分配好所需要的内存。在系统运行过程中不再进行内存的分配/释放，这样就避免了因内存的分配释放带来的总是。而且在系统初始化，因为没有内存碎片，对于大的内存块的需求容易满足。对于其它的应用，在运行时进行动态内存分配。尤其是某些应用所要求的大量固定尺寸的小内存块，这时就可以采用一次分配多次使用的内存分配方案。下面详细介绍这种内存分配方案及其应用场合。 