|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | **linux的物理内存与线性地址空间布局** | | |  | | --- | |  | | |  | |
| |  | | --- | |  | | |  | | --- | | 在支持MMU的32位处理器平台上，Linux系统中的物理存储空间和虚拟存储空间的地址范围分别都是从0x00000000到0xFFFFFFFF，共4GB，但物理存储空间与虚拟存储空间布局完全不同。Linux运行在虚拟存储空间，并负责把系统中实际存在的远小于4GB的物理内存根据不同需求映射到整个4GB的虚拟存储空间中。  **1.物理存储空间布局**  Linux的物理存储空间布局与处理器相关，详细情况可以从处理器用户手册的存储空间分布表（memory map）相关章节中查到，我们这里只列出嵌入式处理器平台Linux物理内存空间的一般布局，如图18-4所示。    图18-4  Linux物理内存空间一般布局示意图  说明：  1）最大node号n不能大于MAX\_NUMNODES-1。  2）MAX\_NUMNODES表示系统支持的最多node数。在ARM系统中，Sharp芯片最多支持16个nodes，其他芯片最多支持4个nodes。  3）numnodes是当前系统中实际的内存node数。  4）在不支持CONFIG\_DISCONTIGMEM选项的系统中，只有一个内存node。  5）最大bank号m不能大于NR\_BANKS-1。  6）NR\_BANKS表示系统中支持的最大内存bank数，一般等于处理器的RAM片选数。在ARM系统中，Sharp芯片最多支持16个banks，其他芯片最多支持8个banks。  7）mem\_init()函数会将所有节点的页帧位码表所占空间、孔洞页描述符空间及空闲内存页都释放掉。  **2.虚拟存储空间布局**  在支持MMU的系统中，当系统做完硬件初始化后就使能MMU功能，这样整个系统就运行在虚拟存储空间中，实现虚拟存储空间到物理存储空间映射功能的是处理器的MMU，而虚拟存储空间与5路存储空间的映射关系则是由Linux内核来管理的。32位系统中物理存储空间占4GB空间，虚拟存储空间同样占4GB空间，Linux把物理空间中实际存在的远远小于4GB的内存空间映射到整个4GB虚拟存储空间中除映射I/O空间之外的全部空间，所以虚拟内存空间远远大于物理内存空间，这就说同一块物理内存可能映射到多处虚拟内存地址空间上，这正是Linux内存管理职责所在。图18-5列出了Linux内核中虚拟内存空间的一般布局（其实I/O空间也在其中，通常占用高端内存空间，在此未标出）。    图18-5  Linux系统虚拟内存空间一般布局示意图  说明：  1）线性地址空间：是指Linux系统中从0x00000000到0xFFFFFFFF整个4GB虚拟存储空间。  2）内核空间：内核空间表示运行在处理器最高级别的超级用户模式（supervisor mode）下的代码或数据，内核空间占用从0xC0000000到0xFFFFFFFF的1GB线性地址空间，内核线性地址空间由所有进程共享，但只有运行在内核态的进程才能访问，用户进程可以通过系统调用切换到内核态访问内核空间，进程运行在内核态时所产生的地址都属于内核空间。  3）用户空间：用户空间占用从0x00000000到0xBFFFFFFF共3GB的线性地址空间，每个进程都有一个独立的3GB用户空间，所以用户空间由每个进程独有，但是内核线程没有用户空间，因为它不产生用户空间地址。另外子进程共享（继承）父进程的用户空间只是使用与父进程相同的用户线性地址到物理内存地址的映射关系，而不是共享父进程用户空间。运行在用户态和内核态的进程都可以访问用户空间。  4）内核逻辑地址空间：是指从PAGE\_OFFSET(3G)到high\_memory(物理内存的大小，最大896)之间的线性地址空间，是系统物理内存映射区，它映射了全部或部分（如果系统包含高端内存）物理内存。内核逻辑地址空间与图18-4中的系统RAM内存物理地址空间是一一对应的（包括内存孔洞也是一一对应的），内核逻辑地址空间中的地址与RAM内存物理地址空间中对应的地址只差一个固定偏移量（3G），如果RAM内存物理地址空间从0x00000000地址编址，那么这个偏移量就是PAGE\_OFFSET。  5）低端内存：内核逻辑地址空间所映射物理内存就是低端内存(实际物理内存的大小，但是小于896)，低端内存在Linux线性地址空间中始终有永久的一一对应的内核逻辑地址，系统初始化过程中将低端内存永久映射到了内核逻辑地址空间，为低端内存建立了虚拟映射页表。低端内存内物理内存的物理地址与线性地址之间的转换可以通过\_\_pa(x)和\_\_va(x)两个宏来进行，#define\_\_pa(x) ((unsigned long)(x)-PAGE\_OFFSET) \_\_pa(x)将内核逻辑地址空间的地址x转换成对应的物理地址，相当于\_\_virt\_to\_phys((unsigned long)(x))，\_\_va(x)则相反，把低端物理内存空间的地址转换成对应的内核逻辑地址，相当于((void \*)\_\_phys\_to\_virt((unsigned long)(x)))。  6）高端内存：低端内存地址之上的物理内存是高端内存（物理内存896之上），高端内存在Linux线性地址空间中没有没有固定的一一对应的内核逻辑地址，系统初始化过程中不会为这些内存建立映射页表将其固定映射到Linux线性地址空间，而是需要使用高端内存的时候才为分配的高端物理内存建立映射页表，使其能够被内核使用，否则不能被使用。高端内存的物理地址于现行地址之间的转换不能使用上面的\_\_pa(x)和\_\_va(x)宏。  7）高端内存概念的由来：如上所述，Linux将4GB的线性地址空间划分成两部分，从0x00000000到0xBFFFFFFF共3GB空间作为用户空间由用户进程独占，这部分线性地址空间并没有固定映射到物理内存空间上；从0xC0000000到0xFFFFFFFF的第4GB线性地址空间作为内核空间，在嵌入式系统中，这部分线性地址空间除了映射物理内存空间之外还要映射处理器内部外设寄存器空间等I/O空间。0xC0000000~high\_memory之间的内核逻辑地址空间专用来固定映射系统中的物理内存，也就是说0xC0000000~high\_memory之间空间大小与系统的物理内存空间大小是相同的（当然在配置了CONFIG\_DISCONTIGMEMD选项的非连续内存系统中，内核逻辑地址空间和物理内存空间一样可能存在内存孔洞），如果系统中的物理内存容量远小于1GB，那么内核现行地址空间中内核逻辑地址空间之上的high\_memory~0xFFFFFFFF之间还有足够的空间来固定映射一些I/O空间。可是，如果系统中的物理内存容量（包括内存孔洞）大于1GB，那么就没有足够的内核线性地址空间来固定映射系统全部物理内存以及一些I/O空间了，为了解决这个问题，在x86处理器平台设置了一个经验值：896MB，就是说，如果系统中的物理内存（包括内存孔洞）大于896MB，那么将前896MB物理内存固定映射到内核逻辑地址空间0xC0000000~0xC0000000+896MB（=high\_memory）上，而896MB之后的物理内存则不建立到内核线性地址空间的固定映射，这部分内存就叫高端物理内存。此时内核线性地址空间high\_memory~0xFFFFFFFF之间的128MB空间就称为高端内存线性地址空间，用来映射高端物理内存和I/O空间。896MB是x86处理器平台的经验值，留了128MB线性地址空间来映射高端内存以及I/O地址空间，我们在嵌入式系统中可以根据具体情况修改这个阈值，比如，MIPS中将这个值设置为0x20000000B（512MB），那么只有当系统中的物理内存空间容量大于0x20000000B时，内核才需要配置CONFIG\_HIGHMEM选项，使能内核对高端内存的分配和映射功能。什么情况需要划分出高端物理内存以及高端物理内存阈值的设置原则见上面的内存页区（zone）概念说明。  8）高端线性地址空间：从high\_memory到0xFFFFFFFF之间的线性地址空间属于高端线性地址空间，其中VMALLOC\_START~VMALLOC\_END之间线性地址被vmalloc()函数用来分配物理上不连续但线性地址空间连续的高端物理内存，或者被vmap()函数用来映射高端或低端物理内存，或者由ioremap()函数来重新映射I/O物理空间。PKMAP\_BASE开始的LAST\_PKMAP（一般等于1024）页线性地址空间被kmap()函数用来永久映射高端物理内存。FIXADDR\_START开始的KM\_TYPE\_NR\*NR\_CPUS页线性地址空间被kmap\_atomic()函数用来临时映射高端物理内存，其他未用高端线性地址空间可以用来在系统初始化期间永久映射I/O地址空间。  Linux 2.6.10内核中的ARM处理器平台部分没有对高端内存的支持，图18-6和图18-7分别列出了SA1100和IXP4XX处理器平台的Linux线性地址空间布局。  **Linux内存线性地址空间大小为4GB，分为2个部分：用户空间部分（通常是3G）和内核空间部分（通常是1G）。在此我们主要关注内核地址空间部分。**    内核通过内核页全局目录来管理所有的物理内存，由于线形地址前3G空间为用户使用，内核页全局目录前768项（刚好3G）除0、1两项外全部为0，后256项（1G）用来管理所有的物理内存。内核页全局目录在编译时静态地定义为swapper\_pg\_dir数组，该数组从物理内存地址0x101000处开始存放。  由图可见， (1) 内核线形地址空间部分从PAGE\_OFFSET（通常定义为3G）开始，为了将内核装入内存，从PAGE\_OFFSET开始8M线形地址用来映射内核所在的物理内存地址； (2)接下来是mem\_map数组，mem\_map的起始线形地址与体系结构相关，比如对于UMA结构，由于从PAGE\_SIZE开始16M线形地址空间对应的16M物理地址空间是DMA区，mem\_map数组通常开始于PAGE\_SIZE+16M的线形地址； (3)从PAGE\_SIZE开始到VMALLOC\_START – VMALLOC\_OFFSET的线形地址空间直接映射到物理内存空间（一一对应影射，物理地址=线形地址-PAGE\_OFFSET），这段区域的大小和机器实际拥有的物理内存大小有关，这儿VMALLOC\_OFFSET在x86上为8M，主要用来防止越界错误； (4)在内存比较小的系统上，余下的线形地址空间（还要再减去空白区即VMALLOC\_OFFSET）被vmalloc()函数用来把不连续的物理地址空间映射到连续的线形地址空间上，在内存比较大的系统上，vmalloc()使用从VMALLOC\_START到VMALLOC\_END（也即PKMAP\_BASE减去2页的空白页大小PAGE\_SIZE）的线形地址空间  (5)此时余下的线形地址空间（还要再减去2页的空白区即VMALLOC\_OFFSET）又可以分成2部分： 第一部分从PKMAP\_BASE到FIXADDR\_START用来由kmap()函数映射高端内存； 第二部分，从FIXADDR\_START到FIXADDR\_TOP，这是一个固定大小的线形地址空间，（引用：Fixed virtual addresses are needed for subsystems that need to know the virtual address at compile time such as the APIC），在x86体系结构上，FIXADDR\_TOP被静态定义为0xFFFFE000,此时这个固定大小空间结束于整个线形地址空间最后4K前面，该固定大小空间大小是在编译时计算出来并存储在\_\_FIXADDR\_SIZE变量中。  正是由于vmalloc()使用区、kmap()使用区及固定大小区的存在才使ZONE\_NORMAL区大小受到限制，由于内核在运行时需要这些函数，因此在线形地址空间中至少要VMALLOC\_RESERVE大小的空间。VMALLOC\_RESERVE的大小与体系结构相关，在x86上，VMALLOC\_RESERVE定义为128M，这就是为什么我们看到ZONE\_NORMAL大小通常是16M到896M的原因。  1、  kmalloc()是内核中最常见的内存分配方式，它最终调用伙伴系统的\_\_get\_free\_pages()函数分配，根据传递给这个函数的flags参数，决定这个函数的分配适合什么场合，如果标志是GFP\_KERNEL则仅仅可以用于进程上下文中，如果标志GFP\_ATOMIC则可以用于中断上下文或者持有锁的代码段中。  kmalloc返回的线形地址是直接映射的，而且用连续物理页满足分配请求，且内置了最大请求数（2\*\*5=32页）。  2、  kmap()是主要用在高端存储器页框的内核映射中，一般是这么使用的: 使用alloc\_pages()在高端存储器区得到struct page结构，然后调用kmap(struct \*page)在内核地址空间PAGE\_OFFSET+896M之后的地址空间中（PKMAP\_BASE到FIXADDR\_STAR）建立永久映射(如果page结构对应的是低端物理内存的页，该函数仅仅返回该页对应的虚拟地址) kmap()也可能引起睡眠，所以不能用在中断和持有锁的代码中  不过kmap 只能对一个物理页进行分配，所以尽量少用。   3、  vmalloc优先使用高端物理内存，但性能上会打些折扣。  vmalloc分配的物理页不会被交换出去;  vmalloc返回的虚地址大于(PAGE\_OFFSET + SIZEOF(phys memory) + GAP)，为VMALLOC\_START----VMALLOC\_END之间的线形地址;  vmalloc使用的是vmlist链表，与管理用户进程的vm\_area\_struct要区别,而后者会swapped；    4、  使用kmap的原因： 对于高端物理内存(896M之后)，并没有和内核地址空间建立一一对应的关系(即虚拟地址=物理地址+PAGE\_OFFSET这样的关系)，所以不能使用get\_free\_pages()这样的页分配器进行内存的分配，而必须使用alloc\_pages()这样的伙伴系统算法的接口得到struct \*page结构，然后将其映射到内核地址空间 | | |