Q：大家可以使用我的 BiscuitOS 项目制作任何版本的 Linux，实现在线调试内核

A: x86还是arm? 跟qemu差不多功能吗？

Q：运行在 qemu 上

A: 回头我先去看个简介

Q: @leon.lee 明天TC就多帮我一下，我回家就弄这个东西了，我还有点TC的问题

给大家分享一个问题的解决办法，刚刚有个朋友问了一个关于如果确保物理内存不被内核使用的问题。

就是我跟他的聊天。

A: 如何制作linux 还有在线调试，有文档吗？

Q：你想调试哪个版本的内核？

Linux 0.x 1.x 2.x 3.x 4.x 5.x 内核制作在线调试文档，在这里

<https://biscuitos.github.io/blog/Kernel_Build/#Linux_0X>

A: 最近工作时接触到的是4.9了，倒没有纠结多少版本，只是为了学习，主要是想知道如何去制作和在线调试. 嗯嗯好的，非常感谢.

内容正是开始：

Q：大家好，一会我要给大家讲一下内存的相关预备知识，大家有问题先提一下，我一会整理回答一下。

A: 了解内存相关预备知识之前还需要了解啥吗

A: 我的问题:从linux 内核启动的时候，到正常支持应用程序运行。 内存分配的经历。

A:

|  |
| --- |
| 1、Linux内核从加电开始到启动完毕之后这段期间是怎么做的  2、简单说下slab的流程  3、内存模块的几个表  4、虚拟内存是怎么回事 |

大概讲讲吧 不用具体到代码 主要是思路上, 直接上手代码的话懵逼了.

Q: 我今天只从框架讲，也就是基础将，后面细节的实践我都会给出文档.

A: 对对 大局上讲讲, 别细节了。。

B: 嗯，先学习框架.

A: 飞机大宝剑已经准备好了 随时准备刷礼物.

C: 从系统到细节，初始化/分配/管理/回收/优化， @BuddyZhang1

D: CMA没搞清楚，能不能讲讲.

E: 我现在比较混沌的是 伙伴系统 页面管理器,slab与slub。

Q:那我开始讲了. 我所讲的内容以后都会图文并茂给大家

A: 开始你的表演 我笔记本准备好了.

Q: 内存管理主要围绕两个东西来讲：虚拟地址和物理地址.

大家开始学习内存管理总会遇到：物理地址，虚拟地址，线性地址，逻辑地址.

内存管理这么多分配器这么多策略，无非就是在为了：虚拟地址和物理地址两个东西。

所以我们今天就从这两个切入。

什么是物理地址，简单说就是 DRAM 的地址。

比如我的 DRAM 的地址是从 0x60000000 开始到 0x80000000,那么 这里所说的地址就是物理地址。

什么是虚拟地址，简单的说就是进程的地址。

32 位系统中，每个进程都拥有 4G 的地址，其中前 3G 是进程的，后 1 G 是内核的，这里所说的地址就是虚拟地址。

什么是线性地址，就是大多数架构中，内存都是平坦模型，就是地址从 0 地址一致到 4G （32 位系统上）。所以这里地址就是线性地址。

什么是逻辑地址？逻辑地址在 X86 上的弄出来的。X86 上面有很多段，每个段由一定的空间，那么 X86 就把虚拟地址加段基地址统称为逻辑地址。

他们几个的关系是：

虚拟地址 <--> 逻辑地址 <--> 线性地址 <--> 物理地址。

随着内核的发展，虚拟地址，逻辑地址，线性地址在 32 位上基本都是一样的概念了。

接下来说说系统是如何从物理地址到虚拟地址建立映射的。

所谓映射就是一个虚拟地址对应一个物理地址。

对应的方式很多，可以通过线性对应，也就是内核低端虚拟地址空间和物理地址的映射方式，也可以通过页表进行映射，也就是进程虚拟地址和物理地址的关系。

系统在启动过程中，会获得物理地址的范围，知道物理地址从什么地方开始，到什么地方结束。

系统获得物理地址之后，会在物理地址上建立全局页目录 swapper\_pg\_dir。

Linux 将虚拟地址/线性地址分为： 低端 3G 是用户空间，高端 1 G 是内核策略。

将物理地址从开始到 1G 结束的部分，通过线性方式和内核虚拟地址一一映射。

也就是内核低端虚拟地址只要进行一下线性处理就可以获得对应的物理地址，而不需要经过页表。

而 3 G 部分的虚拟地址分页策略，在 swapper\_pg\_dir 所在的全局目录建立页表。

swapper\_pg\_dir 是一个数组，虚拟地址就将地址的高 11 位作为索引在 swapper\_pg\_dir 数组中找到 pgd 的入口地址，然后在通过虚拟地址的 中间 10 个作为 pte 的索引，找到也的入口。pte 也是一个数组。最后在将找到的 pte 入口地址的低 12 位一起找到所对应的虚拟地址, 通过这样的方法讲 3 G 的虚拟地址和物理地址挂钩。

通过上面的方法，内核启动初期，虚拟地址就和物理地址建立的映射，其他部分就可以使用这些内存了。

如果你的内存无限大，有上面的映射就可以正常运行内核了，但实际中，内存小的可怜，所以就开始引入了各种内存分配器来管理物理内存和虚拟内存。

管理他们的主要目的就是为了让物理地址或虚拟地址可以分配回收再利用。

这里就开始大家熟知的内存分配器： slub， slabm slob, buddy, PCP, MEMBLOCK, kmalloc,vmalloc,kmap,fixup 等。

但这些内存管理器的主要任务就是维护好物理地址和虚拟地址之间的关系，让自己所管理的部分等到合理的安排。

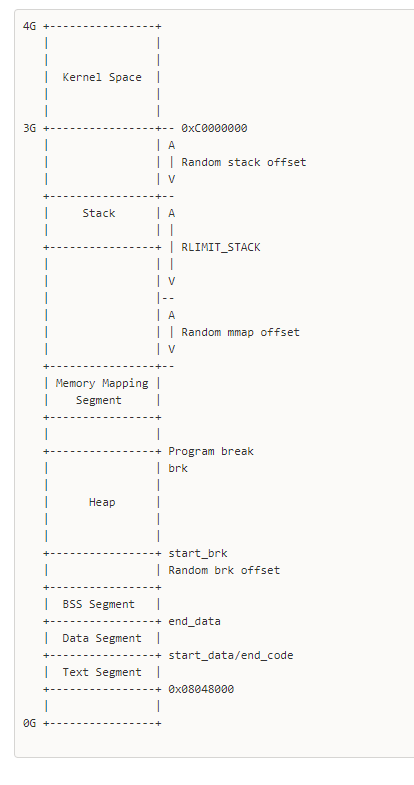
内核初始化之后，就将所有的物理内存交给 MEMBLOKC 管理，系统又将物理内存分配成不同的 ZONE，这里就出现了低端内核，DMA，和高端内存。

他们的出现都是为了：有限的内存空间得到最大的使用。

今天我就将到这里，大家开始提问吧。

Q: 比如我的 DRAM 的地址是从 0x60000000 开始到 0x80000000,那么 这里所说的地址就是物理地址。

A:



DRAM 地址是由 Soc 决定的，你可以看 Soc 手册获得，物理地址也就是 DRAM 在地址总线上的地址，不一定从 0 开始.

Q: 「BuddyZhang1：也就是内核低端虚拟地址只要进行一下线性处理就可以获得对应的物理地址，而不需要经过页表」

- - - - - - - - - - - - - - -

那内核1g还用mmu？

A: 需要的，1G 在以前看来很大，但现在看来很小了，操作系统是动态适应未来的需求，所以在创建之初就要考虑这个问题.

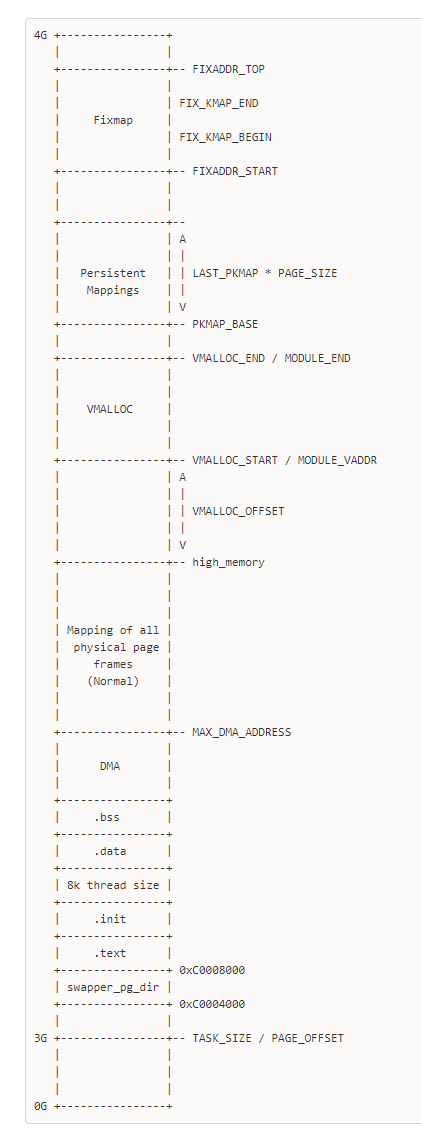
A: 忘记和大家说一点了, 当页表建立好之后，硬件就自动根据页表将虚拟地址装换成物理地址了。

A:什么时候建立页表。这个就是MMU的功能么？

Q: 如果在没有这个硬件的机器上咋整的?

A: X86 文档中提到，可以关闭系统的分页功能，但不能关闭系统的分段功能。但可用通过将每个段的段基地址设置为 0， 段限制设置为 4G，这样就调(跳)过了分段机制.

Q: 虚拟4G地址不同分段内存管理策略和分配策略是怎样的？



这是内核空间采用不同的内存分配器所形成的布局.

可以看出，1G 低端部分采用线性映射，表现比较平坦 。

这位学员，你不要着急，我所讲的以后我都会写成实践文档，让他加都能实践这些内存管理器。今天只是给大家大概说一下。

A:如果大家现在就想切身体会一下内存分配器是如何工作的，我写的第一个专题，也是内核的第一个分配器 MEMBLOCK 物理内存分配器，大家可以照着这个实践，我提供了 60 多个实践文档，详细简绍了原来和实践。

<https://biscuitos.github.io/blog/MMU-ARM32-MEMBLOCK-index/>

Q: 物理内存初始化时，是都放在order 10的链表中吗？就是buddy哪个函数？

A: 现在不能具体给你函数，但后面我会给你的，我记录下来了.

你说的那个函数我几年前好像反复调试过。

Q: 谢谢！你是反复调过，我是反复看过，这就是差距啊。。

A: 最近在研究ARM汇编，为的是写一下 CACHE 和 TLB 的实践方法，有兴趣的童鞋可以和我一起玩.

A: 用户空间到内核空间切换的过程中，页表是如何切换的，以及当时候内存怎么样的变化?

Q: 系统初始化的时候，会建立第一个进程 init\_task，他在的页表指向了 swapper\_pg\_dir 也就是全局页表，内核初始化建立页表的时候，只建立 3G 以上的页表，也就是内核的页表，用户空间的页表不建立。但用户空间执行的时候，fork 调用就会直接copy init\_task 的页表，其他进程也是，所以对所有进程来说，3G 以上的页表都是一致的，所以就有了所有的进程可以看到自己的 0-3G 空间和内核的 1G 空间，而看不到其他进程。看不到其他进行就是看不到其他进程的页表。每个进程建立自己低 3 G 的页表.

A: 请问你在实践的cache类型和层级是？

Q: CMA 可以看看这个 <https://biscuitos.github.io/blog/CMA/>

A: 我记的是 第一次建立页表的时候的1对1的映射16KB 第二次的时候实现的是虚拟映射.

Q: 你所指的是x86还是arm

A:arm

Q: 你说的那个 16KB 是 fixup，那个的页表比较特殊，也是供fixup 分配器使用的。

那个页表的映射和正常页表映射机制不一样，以后我也会讲。

Q: 内存管理就是虚拟地址和物理地址两个东西在不停捣鼓

A: 那缺页中断, 缺页异常, 反向映射.

Q: 缺页就是虚拟地址找不到物理地址就触发一个中断，然后中断就会建立虚拟地址到物理地址的映射.

Q: 我今天只大概讲讲，后面都会把设计的内容总结成专题分享给大家的/

我要把所有分配器都总结成实践专题.



脚本都是我写的，我基本能看出什么问题，这样也节省大家处理 BiscuitOS 问题的时间.

A: 群主这个OS是个操作系统吗？还是什么东西？

Q: 支持各个版本的linux内核/

BiscuitOS 是一个开源项目，用于制作 Linux 0.x ,1.x, 2.x,3.x,4.x,5.x 的可直接运行和在线调试的系统

A: 可以制作各个公司的arm的内核吗？

Q: 可以这么做.

A: dram的地址是soc确定的，也是一种外设，比如是6000000-8000000。soc还有其他外设，占有其他地址，比如说gpio 50000000 这个时候内核去操作的话，分配的虚拟地址空间是内核空间的 3g以上的吧，是不是这样理解？

A: 6000000-8000000是物理地址啊，50000000也是物理地址吗？

Q: 都是物理地址

A: 要经过MMU映射吧, 才会成为虚拟地址?

Q: 进行 ioremap 之后就到内核的虚拟地址了.

A: 我的意思是，映射后占用的虚拟地址是。内核空间的1g吧?

Q: 嗯嗯,但要注意用户空间使用 remap 函数。

那就到虚拟地址空间了。

A: 那你最开始发的图片，是4g虚拟地址的分配吧。

Q: 我发了两张图。提一张是用户空间，第二张是内核空间。

A: 后面可以详细描述一下里面分配吗?

A:我帮群主为新人说明一下本群的意义：了解和学习linux内核运行机制，包括内存管理、进程管理、文件系统、网络等

A:群主，你搭建的系统是基于qemu的吗，我没仔细看.

Q:是的。

A1: 群主有基于buildroot吗？我之前记得buildroot好像也是类似的功能

A2: buildroot就是做跟文件系统，然后build内核之类的动作。不过能支持1.0这么老的内核还是不行。

Q:不是，是我用Kbuild 自己写的。我的项目做的比较单一，就是服务内核调试开发。

BiscuitOS 制作的内核所有版本都支持gdb在线调试。

Q:根文件系统就是一个简单的busybox。

还配套详细的中文教程

<https://biscuitos.github.io/blog/BiscuitOS_Catalogue/>

A: 根文件系统的作用是啥？

Q1: linux 启动要根文件系统,

Q2:提供用户空间运行的简单环境,

Q3:有各种命令和脚本, 不然的话 shell 都没有.

A: 这个根文件系统用的什么文件系统呢？

Q: Ext4

大家好，我说讲解的内容想实践的请看这个网址，我所有的实践内容都整理好放在那里里，大家可以看看.

<https://biscuitos.github.io/blog/BiscuitOS_Catalogue/>

昨天讲的内容我已经写成实践文档，只是对应的专题还没写好，写好分享给大家

A:张老师，上面你给出了虚拟地址分布图，那内核被BIOS/UBOOT加载到物理内存的分布图是什么样的呢？内核启动后物理内存分配图又有什么变化呢？

Q: 这个时候我不太好讲这个话题，不让会和内核的布局混淆。以后有机会再将，但可以通俗的这么认为内核加载到内存之后，uboot/BIOS 把执行权给kernel 之后，整个内存空间除了内核自己，其他东西都没有。然后内核以此为原点开始初始化内存.

Q1:

这个东西我以前见过一个视频教程，讲得很清楚，大家可以参考.

《毕业班第2课第2.3节u-boot分析启动过程之重定位》。