Kernel Hugpage intro

David Zhou

这章简单讲讲hugepage的实现，当我们想要使用hugepage的时候有两种方式：

1. 设置kernel parameters

2. 运行时通过echo xxx > /proc/sys/vm/nr\_hugepages 或其他文件写入, xxx是需要的hugepage个数。

那么这两种方式有什么区别？内核又是怎么实现hugepage的分配呢？分配的时候内存做了些什么操作？

接下来让我们进入内核，分析一下上述几个问题。

先看第一种情况，我们在kernel boot阶段设置boot parameter：hugepagesz=2M hugepages=5462 transparent\_hugepage=never

Kernel提供一个宏\_\_setup:

#define \_\_setup\_param(str, unique\_id, fn, early) \

static const char \_\_setup\_str\_##unique\_id[] \_\_initconst \

\_\_aligned(1) = str; \

static struct obs\_kernel\_param \_\_setup\_##unique\_id \

\_\_used \_\_section(".init.setup") \

\_\_attribute\_\_((aligned((sizeof(long))))) \

= { \_\_setup\_str\_##unique\_id, fn, early }

#define \_\_setup(str, fn) \

\_\_setup\_param(str, fn, fn, 0)

可以看到凡是用\_\_setup宏定义的内容都会放到.init.setup段里面。那么这个段是做什么的呢？我们继续来看链接文件上的描述：

.init.data : {

. = ALIGN(16); \_\_setup\_start = .; KEEP(\*(.init.setup)) \_\_setup\_end = .;

}

Kernel会用\_\_setup\_start和\_\_setup\_end两个变量来表示这一块区域的开始和结束。现在就清楚了，kernel是在下面函数调用的。至于command\_line怎么来的，请参考《K3 uboot.docx》

parse\_args("Booting kernel",

static\_command\_line, \_\_start\_\_\_param,

\_\_stop\_\_\_param - \_\_start\_\_\_param,

-1, -1, NULL, &unknown\_bootoption);

调用顺序：

+----------------+ +---------------------+ +--------------------+

| parse\_args | -> | unknown\_boot\_option | -> |obsolete\_checksetup |

+----------------+ +---------------------+ +--------------------+

最终在obsolet\_checksetup里面调用参数注册的函数。先分析一下parse\_args的过程：

char \*parse\_args()

{

char \*param, \*val, \*err = NULL;

args = skip\_spaces(args);

if (\*args)

pr\_debug("doing %s, parsing ARGS: '%s'\n", doing, args);

while (\*args) {

int ret;

int irq\_was\_disabled;

args = next\_arg(args, &param, &val);

if (!val && strcmp(param, "--") == 0)

return err ?: args;

irq\_was\_disabled = irqs\_disabled();

ret = parse\_one(param, val, doing, params, num,

min\_level, max\_level, arg, unknown);

if (irq\_was\_disabled && !irqs\_disabled())

pr\_warn("%s: option '%s' enabled irq's!\n",

doing, param);

switch (ret) {

case 0:

continue;

case -ENOENT:

pr\_err("%s: Unknown parameter `%s'\n", doing, param);

break;

case -ENOSPC:

pr\_err("%s: `%s' too large for parameter `%s'\n",

doing, val ?: "", param);

break;

default:

pr\_err("%s: `%s' invalid for parameter `%s'\n",

doing, val ?: "", param);

break;

}

err = ERR\_PTR(ret);

}

return err;

}

首先剔除前面的空格，然后拿到第一个参数next\_arg(args, &param, &val);参数形式param=value。解析后的param赋值给param，value赋值给val。然后进入parse\_one函数进行进一步分析：

static int parse\_one()

{

unsigned int i;

int err;

for (i = 0; i < num\_params; i++) {

if (parameq(param, params[i].name)) {

if (params[i].level < min\_level

|| params[i].level > max\_level)

return 0;

/\* No one handled NULL, so do it here. \*/

if (!val &&

!(params[i].ops->flags & KERNEL\_PARAM\_OPS\_FL\_NOARG))

return -EINVAL;

pr\_debug("handling %s with %p\n", param,

params[i].ops->set);

kernel\_param\_lock(params[i].mod);

if (param\_check\_unsafe(&params[i]))

err = params[i].ops->set(val, &params[i]);

else

err = -EPERM;

kernel\_param\_unlock(params[i].mod);

return err;

}

}

if (handle\_unknown) {

pr\_debug("doing %s: %s='%s'\n", doing, param, val);

return handle\_unknown(param, val, doing, arg);

}

pr\_debug("Unknown argument '%s'\n", param);

return -ENOENT;

}

我们先忽略for循环这部分，这部分是以core\_param或者module\_param\_string等宏声明的参数进行处理的，主要是用于设置传入的参数值。处理函数是ops下面的set函数。

然后调用handle\_unknown函数，这个函数是parse\_args函数传入的unkown\_bootoption，上面的函数调用关系得到最终调用obsolete\_checksetup函数。

这个函数就不多介绍了，主要工作就是从\_\_setup\_start开始，比较两个param是否相等，是否有默认处理函数，如果有，那就执行处理函数。

有了这部分基础以后，我们来看hugepage的命令行参数：hugepagesz、hugepages、transparent\_hugepage，总共3个参数。

\_\_setup("hugepagesz=", hugepagesz\_setup);

\_\_setup("hugepages=", hugepages\_setup);

\_\_setup("transparent\_hugepage=", setup\_transparent\_hugepage);

在分析这几个函数之前，我们先来简单分析一下linux memory的管理，特别是伙伴系统管理，详细简析请参考《K3 kernel.docx》。

从《K3 uboot.docx》我们知道了Linux所用的内存是怎么来的了。接下来就是处理所得到的内存了。

+--------------+ +-------------------+ +------------------+

| setup\_arch | -> | setup\_machine\_fdt | -> |early\_init\_dt\_scan| ->

+--------------+ +-------------------+ +------------------+

+------------------------+ +---------------------------+

|early\_init\_dt\_scan\_nodes| -> | early\_init\_dt\_scan\_memory |

+------------------------+ +---------------------------+

在early\_init\_dt\_scan\_memory里面从dts里面取出memory字段，调用early\_init\_dt\_add\_memory\_arch->memblock\_add把内存添加到memblock结构体里面。