kernel hacker 修炼之道——李万鹏

男儿立志出乡关， 学不成名死不还。 埋骨何须桑梓地， 人生无处不青山。 ——西乡隆盛诗

kernel hacker修炼之道之PCI subsystem(六)

分类： linux内核编程 PCI-e/PCI 2012-02-26 18:39 1072人阅读 评论(3) 收藏 举报

kernel hacker修炼之道之PCI subsystem(六)

作者　李万鹏

第二步分析PCI core对PCI device resource的分配，包括：I/O，Memory，在pcibios_init函数中，遍历完PCI tree之后会调用pcibios_resource_survey函数来分配资源：

```
1450void __init pcibios_resource_survey(void)
1451{
1452 struct pci_bus *b;
1453
1454 /* Allocate and assign resources. If we re-assign everything, then
1455 * we skip the allocate phase
1456 */
1457 list_for_each_entry(b, &pci_root_buses, node)
1458 pcibios_allocate_bus_resources(b);
1459
1460 if (!pci_has_flag(PCI_REASSIGN_ALL_RSRC)) {
1461 pcibios_allocate_resources(0);
1462 pcibios_allocate_resources(1);
1463 }
1464
1465 /* Before we start assigning unassigned resource, we try to reserve
1466 * the low IO area and the VGA memory area if they intersect the
1467 * bus available resources to avoid allocating things on top of them
1468 */
1469 if (!pci_has_flag(PCI_PROBE_ONLY)) {
1470 list_for_each_entry(b, &pci_root_buses, node)
1471 pcibios_reserve_legacy_regions(b);
1472 }
1473
1474 /* Now, if the platform didn't decide to blindly trust the firmware,
1475 * we proceed to assigning things that were left unassigned
1476 */
1477 if (!pci_has_flag(PCI_PROBE_ONLY)) {
1478 pr_debug("PCI: Assigning unassigned resources...\n");
1479 pci_assign_unassigned_resources();
1480 }
1481
1482 /* Call machine dependent fixup */
1483 if (ppc_md.pcibios_fixup)
1484 ppc_md.pcibios_fixup();
1485}
```

首先给各个bus分配resource，然后再给挂在bus上的各个device分配resource，然后对未成功分配到resource的device进行重新分配。pci_probe_only是PowerPC_64下的一个全局变量，如果这个全局变量为1，那么pci_flags的bit field PCI_PROBE_ONLY将会被设置为1。如果PCI_PROBE_ONLY标志被设置，则一些resource没有成功分配的设备不会被PCI core重新分配。这个是与平台有关的，有一个叫“powernv”，它的resource是由平台相关的代码分配的，而一些平台上，如果PCI_PROBE_ONLY标志没有被设置，则由PCI core分配。

给bus分配resource是一个递归的过程：

```
1253void pcibios_allocate_bus_resources(struct pci_bus *bus)
1254{
1255 struct pci_bus *b;
1256 int i;
1257 struct resource *res, *pr;
1258
1259 pr_debug("PCI: Allocating bus resources for %04x:%02x...\n",
1260 pci_domain_nr(bus), bus->number);
1261
1262 pci_bus_for_each_resource(bus, res, i) {
1263 if (!res || !res->flags || res->start > res->end || res->parent)
1264 continue;
1265 if (bus->parent == NULL)
1266 pr = (res->flags & IORESOURCE_IO) ?
1267 &ioport_resource : &iomem_resource;
1268 else {
1269 /* Don't bother with non-root busses when
1270 * re-assigning all resources. We clear the
1271 * resource flags as if they were colliding
1272 * and as such ensure proper re-allocation
1273 * later.
1274 */
1275 if (pci_has_flag(PCI_REASSIGN_ALL_RSRC))
1276 goto clear_resource;
1277 pr = pci_find_parent_resource(bus->self, res);
1278 if (pr == res) {
1279 /* this happens when the generic PCI
1280 * code (wrongly) decides that this
1281 * bridge is transparent -- paulus
1282 */
1283 continue;
1284 }
1285 }
1286
1287 pr_debug("PCI: %s (bus %d) bridge rsrc %d: %016llx-%016llx "
1288 "[0x%x], parent %p (%s)\n",
1289 bus->self ? pci_name(bus->self) : "PHB",
1290 bus->number, i,
1291 (unsigned long long)res->start,
1292 (unsigned long long)res->end,
1293 (unsigned int)res->flags,
1294 pr, (pr && pr->name) ? pr->name : "nil");
1295
1296 if (pr && !(pr->flags & IORESOURCE_UNSET)) {
1297 if (request_resource(pr, res) == 0)
1298 continue;
1299 /*
1300 * Must be a conflict with an existing entry.
1301 * Move that entry (or entries) under the
1302 * bridge resource and try again.
1303 */
```

```
1304 if (reparent_resources(pr, res) == 0)
1305 continue;
1306 }
1307 printk(KERN_WARNING "PCI: Cannot allocate resource region "
1308 "%d of PCI bridge %d, will remap\n", i, bus->number);
1309clear_resource:
1310 res->start = res->end = 0;
1311 res->flags = 0;
1312 }
1313
1314 list_for_each_entry(b, &bus->children, node)
1315 pcibios_allocate_bus_resources(b);
```

1316}如果bus是各个domain的toplevel bus，则他们的父resource就是ioport_resource和iomem_resource，在kernel/resource.c文件中定义的。

```
25struct resource ioport_resource = {
26 .name = "PCI IO",
27 .start = 0,
28 .end = IO_SPACE_LIMIT,
29 .flags = IORESOURCE_IO,
30};
31EXPORT_SYMBOL(ioport_resource);
32
33struct resource iomem_resource = {
34 .name = "PCI mem",
35 .start = 0,
36 .end = -1,
37 .flags = IORESOURCE_MEM,
38};
```

如果只是普通设备，则在bus的parent bus(也就是引出这个bus的bridge的primary bus)上寻找resource，会调用pci_find_parent_resource函数，如下：

```
444pci_find_parent_resource(const struct pci_dev *dev, struct resource *res)
445{
446 const struct pci_bus *bus = dev->bus;
447 int i;
448 struct resource *best = NULL, *r;
449
450 pci_bus_for_each_resource(bus, r, i) {
451 if (!r)
452 continue;
453 if (res->start && !(res->start >= r->start && res->end <= r->end))
454 continue; /* Not contained */
455 if ((res->flags ^ r->flags) & (IORESOURCE_IO | IORESOURCE_MEM))
456 continue; /* Wrong type */
457 if (!((res->flags ^ r->flags) & IORESOURCE_PREFETCH))
458 return r; /* Exact match */
459 /* We can't insert a non-prefetch resource inside a prefetchable parent .. */
460 if (r->flags & IORESOURCE_PREFETCH)
461 continue;
462 /* .. but we can put a prefetchable resource inside a non-prefetchable one */
463 if (!best)
464 best = r;
465 }
466 return best;
```

467}这里必须满足几个条件才算在parent bus中找到了resource：

- 请求的resource range必须在parent bus的resource range之内
- resource的标志必须与parent bus的匹配
- 如果parent bus的resource是prefetchable的，而请求的是prefetchable or unprefetchable都可以

- 如果parent bus的resource是unprefetchable，而请求的是prefetchable则不可以

如果在parent bus中找到了合适的resource并且没有设置IORESOURCE_UNSET标志，则调用request_resource函数：

```
254int request_resource(struct resource *root, struct resource *new)
255{
256 struct resource *conflict;
257
258 conflict = request_resource_conflict(root, new);
259 return conflict ? -EBUSY : 0;
260}
```

这里调用了request_resource_conflict函数：

```
237struct resource *request_resource_conflict(struct resource *root, struct resource *new)
238{
239 struct resource *conflict;
240
241 write_lock(&resource_lock);
242 conflict = __request_resource(root, new);
243 write_unlock(&resource_lock);
244 return conflict;
245}
```

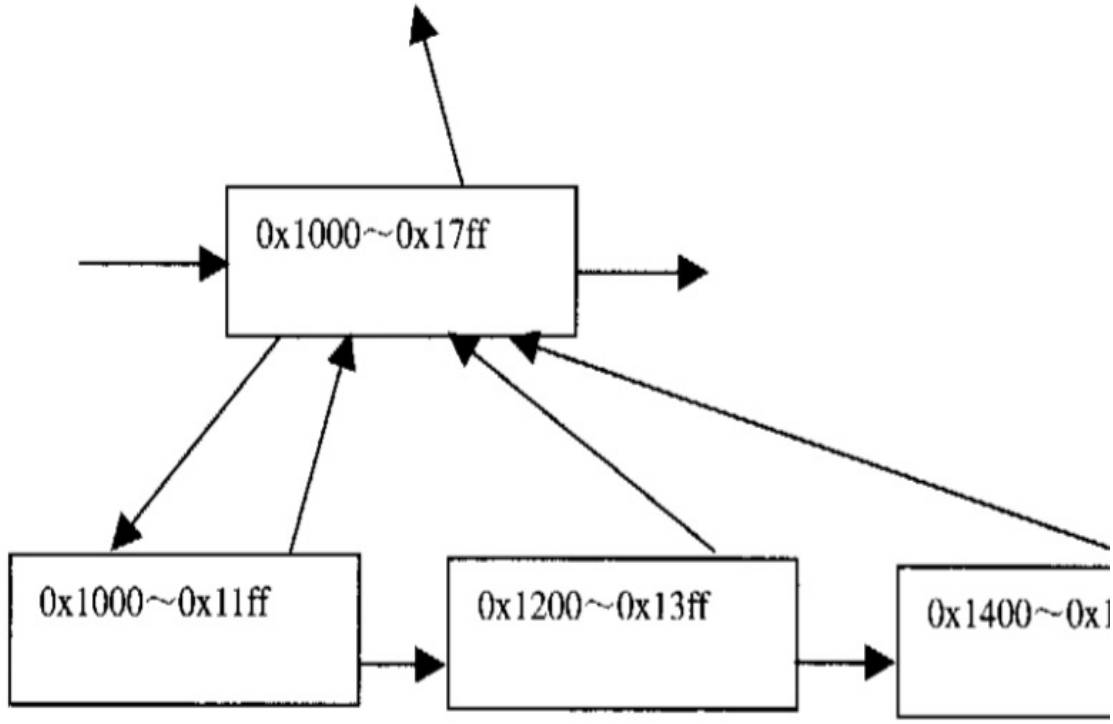
最终调用到核心的__request_resource函数：

```
153static struct resource * __request_resource(struct resource *root, struct resource *new)
154{
155 resource_size_t start = new->start;
156 resource_size_t end = new->end;
157 struct resource *tmp, **p;
158
159 if (end < start)
160 return root;
161 if (start < root->start)
162 return root;
163 if (end > root->end)
164 return root;
165 p = &root->child;
166 for (;;) {
167 tmp = *p;
168 if (!tmp || tmp->start > end) {
169 new->sibling = tmp;
170 *p = new;
171 new->parent = root;
172 return NULL;
173 }
174 p = &tmp->sibling;
175 if (tmp->end < start)
176 continue;
177 return tmp;
178 }
179}
```

要看懂这段代码必须分析内核中的resource结构，定义在include/linux

```
18struct resource {
19        resource_size_t start;
20        resource_size_t end;
21        const char *name;
22        unsigned long flags;
23        struct resource *parent, *sibling, *child;
24};
```

这里的start,end分别是resource的起始地址和结束地址。name是re



__request_resource函数主要是将请求的resource添加到parent resouce的child链表上，并将reousrce的parent指针指向parent resource。所以后面对未分配的resource重新分配的时候会用parent指针是否为空来判断resource是否被分配。

下面是为device分配资源的部分：

```
1346static void __init pcibios_allocate_resources(int pass)
1347{
1348 struct pci_dev *dev = NULL;
1349 int idx, disabled;
1350 u16 command;
1351 struct resource *r;
1352
1353 for_each_pci_dev(dev) {
1354 pci_read_config_word(dev, PCI_COMMAND, &command);
1355 for (idx = 0; idx <= PCI_ROM_RESOURCE; idx++) {
1356 r = &dev->resource[idx];
1357 if (r->parent) /* Already allocated */
1358 continue;
1359 if (!r->flags || (r->flags & IORESOURCE_UNSET))
1360 continue; /* Not assigned at all */
```

```
1361 /* We only allocate ROMs on pass 1 just in case they
1362 * have been screwed up by firmware
1363 */
1364 if (idx == PCI_ROM_RESOURCE )
1365 disabled = 1;
1366 if (r->flags & IORESOURCE_IO)
1367 disabled = !(command & PCI_COMMAND_IO);
1368 else
1369 disabled = !(command & PCI_COMMAND_MEMORY);
1370 if (pass == disabled)
1371 alloc_resource(dev, idx);
1372 }
1373 if (pass)
1374 continue;
1375 r = &dev->resource[PCI_ROM_RESOURCE];
1376 if (r->flags) {
1377 /* Turn the ROM off, leave the resource region,
1378 * but keep it unregistered.
1379 */
1380 u32 reg;
1381 pci_read_config_dword(dev, dev->rom_base_reg, ®);
1382 if (reg & PCI_ROM_ADDRESS_ENABLE) {
1383 pr_debug("PCI: Switching off ROM of %s\n",
1384 pci_name(dev));
1385 r->flags &= ~IORESOURCE_ROM_ENABLE;
1386 pci_write_config_dword(dev, dev->rom_base_reg,
1387 reg & ~PCI_ROM_ADDRESS_ENABLE);
1388 }
1389 }
1390 }
```

1391}这里对6个resource，I/O，MMIO，prefetch MMIO进行分配，这6个资源如果在第一趟也就是pass 0的时候如果是生效的就在pass 0的时候分配，否则在pass 1的时候分配额。还有另一个资源ROM，ROM如果已由firmware分配，则在pass 0将其关闭，如果驱动程序需要用的时候再打开。否则，如果ROM没有被firmware分配(r->parent == NULL)，则在第二趟的时候分配。

```
1318static inline void __devinit alloc_resource(struct pci_dev *dev, int idx)
1319{
1320 struct resource *pr, *r = &dev->resource[idx];
1321
1322 pr_debug("PCI: Allocating %s: Resource %d: %016llx..%016llx [%x]\n",
1323 pci_name(dev), idx,
1324 (unsigned long long)r->start,
1325 (unsigned long long)r->end,
1326 (unsigned int)r->flags);
1327
1328 pr = pci_find_parent_resource(dev, r);
1329 if (!pr || (pr->flags & IORESOURCE_UNSET) ||
1330 request_resource(pr, r) < 0) {
1331 printk(KERN_WARNING "PCI: Cannot allocate resource region %d"
1332 " of device %s, will remap\n", idx, pci_name(dev));
1333 if (pr)
1334 pr_debug("PCI: parent is %p: %016llx-%016llx [%x]\n",
1335 pr,
1336 (unsigned long long)pr->start,
1337 (unsigned long long)pr->end,
1338 (unsigned int)pr->flags);
1339 /* We'll assign a new address later */
1340 r->flags |= IORESOURCE_UNSET;
1341 r->end -= r->start;
1342 r->start = 0;
1343 }
1344}
```

如果device所在的bus有资源，则调用request_resource函数进行分配。否则

如果device所在的bus没有这个resource或resource的标志是IORESOURCE_UNSET，或者request_resource冲突，则等以后重新分配，这里设置了r->flags，r->end，r->start。如果分配失败会打印PCI: Cannot allocate resource region xx of device xxxxx, will remap如下：

```
PCI: Allocating 0000:01:00.0: Resource 0: 00000000ff7f4000..00000000ff7f4007 [20101]
pci_find_parent_resource 00000000ff7ec000..00000000ff7fbfff
PCI: Allocating 0000:01:00.0: Resource 1: 00000000ff7f4040..00000000ff7f4043 [20101]
pci_find_parent_resource 00000000ff7ec000..00000000ff7fbfff
PCI: Allocating 0000:01:00.0: Resource 2: 00000000ff7f4200..00000000ff7f4207 [20101]
pci_find_parent_resource 00000000ff7ec000..00000000ff7fbfff
PCI: Allocating 0000:01:00.0: Resource 3: 00000000ff7f4800..00000000ff7f4803 [20101]
pci_find_parent_resource 00000000ff7ec000..00000000ff7fbfff
PCI: Allocating 0000:01:00.0: Resource 4: 00000000000ec000..00000000000ec00f [20101]
PCI: Cannot allocate resource region 4 of device 0000:01:00.0, will remap
PCI: Allocating 0000:01:00.0: Resource 5: 0000000000800000..00000000008003ff [20200]
PCI: Cannot allocate resource region 5 of device 0000:01:00.0, will remap
```