Linux

Table des matières

1 Vu d'ensemble

- *include/linux* : headers génériques
- *kernel* : code générique
- *drivers/* : code génériques des drivers
- *include/asm-powerpc/* : headers spécifique au processeur
- *arch/powerpc/kernel/* : headers & code spécifique au processeur
- *arch/powerpc/platforms/cesrio/* : headers & code spécifique

2 Les points d'entrees

fichier *include/linux/init.h* :

```
/* These macros are used to mark some functions or
* initialized data (doesn't apply to uninitialized data)
* as 'initialization' functions. The kernel can take this
* as hint that the function is used only during the initialization
* phase and free up used memory resources after
*
* Usage:
* For functions:
*
* You should add __init immediately before the function name
...
/*

#define __init __attribute__ ((__section__ (".init.text"))) __cold
#define subsys_initcall(fn) __define_initcall("4",fn,4)
```

3 Les listes

- *include/linux/pci.h* :

```
struct pci_bus *bus;
...
list_for_each_entry(bus, &pci_root_buses, node)
```

- *include/linux/liste.h* :

```
struct list_head {
  struct list_head *next, *prev;
};

/**
* list_for_each_entry -iterate over list of given type
* @pos: the type * to use as a loop cursor.
* @head: the head for your list.
* @member: the name of the list_struct within the struct.
*/
#define list_for_each_entry(pos, head, member) \
for (pos = list_entry((head)->next, typeof(*pos), member); \
prefetch(pos->member.next), &pos->member != (head);  \
pos = list_entry(pos->member.next, typeof(*pos), member))

#define list_entry(ptr, type, member) \
container_of(ptr, type, member)
```

- *include/linux/kernel.h* : L'opérateur `typeof`, cf ce lien, semble permettre de faire du polymorphisme en C.

```
/**
* container_of - cast a member of a structure out to the containing structure
* @ptr: the pointer to the member.
* @type: the type of the container struct this is embedded in.
* @member: the name of the member within the struct.
*
*/
#define container_of(ptr, type, member) ({ \
  const typeof( ((type *)0)->member ) *__mptr = (ptr); \
  (type *)( (char *)__mptr - offsetof(type,member) );})
```

- *include/linux/prefetch.h* :

```
/*
prefetch(x) attempts to pre-emptively get the memory pointed to
by address "x" into the CPU L1 cache.
...
*/
#ifndef ARCH_HAS_PREFETCH
#define prefetch(x) __builtin_prefetch(x)
#endif
```

- *include/asm-powerpc/processeur.h* :

```
#define ARCH_HAS_PREFETCH
static inline void prefetch(const void *x)
{
```

```
  if (unlikely(!x))
  return;

  __asm__ __volatile__ ("dcbt 0,%0" : : "r" (x));
}
```

- *include/asm-powerpc/processeur.h* :

```
#define ARCH_HAS_PREFETCH
static inline void prefetch(const void *x)
{
  if (unlikely(!x))
  return;

  __asm__ __volatile__ ("dcbt 0,%0" : : "r" (x));
}
```

- *include/asm-powerpc/processeur.h* : The `__builtin_expect` is a method that gcc (versions >= 2.96) offer for programmers to indicate branch prediction information to the compiler. The return value of `__builtin_expect` is the first argument (which could only be an integer) passed to it.
  Cf "info gcc" : celà permet d'informer le compiler lequel des 2 blocks `if` ou `else` est le plus probable.

```
#define likely(x) __builtin_expect(!!(x), 1)
#define unlikely(x) __builtin_expect(!!(x), 0)
```

4 Linux pci

cf cette url.

- *include/linux/pci.h* :

```
/*
* The pci_dev structure is used to describe PCI devices.
*/
struct pci_dev {
  ...
  struct pci_bus *bus; /* bus this device is on */
  ...
}

struct pci_bus {
  struct pci_ops *ops; /* configuration access functions */
  void *sysdata; /* hook for sys-specific extension */
  struct proc_dir_entry *procdir; /* directory entry in /proc/bus/pci */

  unsigned char number; /* bus number */
  char name[48];
};

/* Low-level architecture-dependent routines */

struct pci_ops {
  int (*read)(struct pci_bus *bus, unsigned int devfn, int where, int size, u32 *val);
  int (*write)(struct pci_bus *bus, unsigned int devfn, int where, int size, u32 val);
};
```

- *include/asm-powerpc/pci-bridge.h* :

```
/*
* Structure of a PCI controller (host bridge)
*/
struct pci_controller {
  void *arch_data;
  int first_busno;
  volatile unsigned int __iomem *cfg_addr;
};
```

5 Linux pci initialization

La prise en charge du bus pci demare avec les fichiers :

- *powerpc/platforms/cesrio/early.c* :

```
__init void rio_init_board(unsigned long offset)
{
  ...
  rio_set_cmd_line(offset);
  ...
}

static __init void
rio_set_cmd_line (unsigned long offset)
{
  ...
  CesResourceTable_t * restable;
  restable = (CesResourceTable_t *)cesRT_GetTable();

  /* 1) */
  cesRT_Init(RIO_PTRRELOC(restable), NULL, offset);
  ...
}
```

- *powerpc/platforms/cesrio/pci.c* :

```
/* call by setup_arch(&command_line); from /init/main.c */
void __init rio_find_bridges(void)
{
  // enable slots
  rio_init_pci_slots();

  // enable busses
  for (i = 0; i < RIO_PCI_BUSSES; i++) {
    ...
    cfg->enabled = 1;
  }

  printk("XPC-PCI bridges recognized:");

  /*
  * Configure PCI buses present and enabled.
  */
  for (i = 0; i < RIO_PCI_BUSSS; i++) {
    /* 2) */
    rio_init_pci_bus(i);
  }
  ...
}
```

- *arch/powerpc/kernel/pci_32.c* :

```
static int __init
pcibios_init(void)
{
  printk(KERN_INFO "PCI: Probing PCI hardware\n");
```

```
  /* 3) Build the dev_bus and dev_pci :
  Scan all of the recorded PCI controllers.
  */
  list_for_each_entry_safe(hose, tmp, &hose_list, list_node) {
    bus = pci_scan_bus_parented(hose->parent, hose->first_busno, hose->ops, hose);
  }
  ...
  /* 4)
  Call machine dependent fixup
  cf arch/powerpc/platforms/cesrio/pci.c:473:
  ppc_md.pcibios_fixup = rio_pcibios_fixup
  */
  if (ppc_md.pcibios_fixup)
  ppc_md.pcibios_fixup();
  ...
  /* 5)
  Allocate and assign resources
  un seul appel pour le bus 20
  */
  pcibios_assign_resources();
  ...
}
```

1. Fichier *include/ces/cesResourceTable.h* :

```
/* 1) */
/* cesRT_Init: Retrieve the initial table from hardware.
* This function can be called from virtually any weird context.
* It is guaranted that it does not call into any function outside
* of cesResourceTable.
* A space of 4096 Bytes has to be provided.
* Arguments:
* restable: destination address where the table should be copied to.
* phys_to_virt: a function to convert virtual to physical addresses. A value of NULL means 1:1 mapping
* offset: This is only required if running in a context where the load address differs from the link add
*
* Return value:
* NULL on success
* otherwise: pointer to a string containing the error message
*/
CesError cesRT_Init(CesResourceTable_t *restable, void*(*phys_to_virt)(CesUint32 phys), CesUint32 offset
```

2. fichier *arch/powerpc/platforms/cesrio/pci.c* :

```
/* 2) Access physique aux registres */

static struct pci_ops rio_pci_ops;

void __init rio_init_pci_bus(int i) {
  ...
  hose->ops = &rio_pci_ops;
  hose->first_busno = i << 4;
  ...
}

static struct pci_ops rio_pci_ops = {
  rio_pcibios_read_config,            /* <= pci_bus_read_config_dword */
  rio_pcibios_write_config            /* <= pci_bus_write_config_dword */
};
```

```
int rio_pcibios_read_config(struct pci_bus *bus, unsigned int devfn, int offset,
int len, u32 *val)
{
  ...
  if (hose->first_busno == bus->number &&
  !pci_slot_is_enabled(cfg, devno)) {
    *val = -0;
    return PCIBIOS_DEVICE_NOT_FOUND;
  }
  ...
}

static int pci_slot_is_enabled(rio_pci_cfg_t *cfg, int devno) {
  int id = cfg - rio_pci_cfg;
  if (devno >= pci_slot_count[id])
  return 0;
  return pci_slot_info[id][devno].visible;
}
```

3. Fichier *drivers/pci/probe.c* :

```
/* 3) */
/*
* If it's a bridge, configure it and scan the bus behind it.
* For CardBus bridges, we don't scan behind as the devices will
* be handled by the bridge driver itself.
*
* We need to process bridges in two passes -- first we scan those
* already configured by the BIOS and after we are done with all of
* them, we proceed to assigning numbers to the remaining buses in
* order to avoid overlaps between old and new bus numbers.
*/
int pci_scan_bridge(struct pci_bus *bus, struct pci_dev * dev, int max, int pass)
{
  /* bus number affectation */
  pci_read_config_dword(dev, PCI_PRIMARY_BUS, &buses);

  pr_debug("PCI: Scanning behind PCI bridge %s, config %06x, pass %d\n",
  pci_name(dev), buses & 0xffffff, pass);
}
...

struct pci_bus *pci_scan_bus_parented(struct device *parent,
int bus, struct pci_ops *ops, void *sysdata)
{
  struct pci_bus *b;

  b = pci_create_bus(parent, bus, ops, sysdata);
  if (b)
  b->subordinate = pci_scan_child_bus(b);
  return b;
}
...
/*
* Read the config data for a PCI device, sanity-check it
* and fill in the dev structure...
*/
static struct pci_dev * __devinit
```

```
pci_scan_device(struct pci_bus *bus, int devfn)
{
  ...
      /* if not disabled by 1) or 2) then call rio_pcibios_read_config */

   if (pci_bus_read_config_dword(bus, devfn, PCI_VENDOR_ID, &l)){
pr_debug(":o) %i is disabled\n", devfn);
return NULL;
}

/* some broken boards return 0 or ~0 if a slot is empty: */
if (l == 0xffffffff || l == 0x00000000 ||
    l == 0x0000ffff || l == 0xffff0000){
pr_debug(":o) %i is empty slot\n", devfn);
return NULL;
}
    ...
    dev = alloc_pci_dev();
    ...
    if (pci_setup_device(dev) < 0) {
      kfree(dev);
      return NULL;
    }
    return dev;
  }
```

4. Fichier *arch/powerpc/platforms/cesrio/pci.c* :

```
/* 4) */
static void
__init rio_setup_pci_ptrs(void){
  ...
  ppc_md.pcibios_fixup = rio_pcibios_fixup;
}
...
static void __init rio_pcibios_fixup(void) {
  ...
  if (cpci_full_enum && hose->arch_data == cpci_cfg) {
    printk("%s: cpci_full_enum: Doing a full CPCI enumeration and resource reassignment.\n", __func__);
    printk("%s: cpci_full_enum: Warning: This overrides PPC_Mon resource allocation!\n", __func__);
    ...
}}
```

5. Fichier *arch/powerpc/kernel/pci_32.c* :

```
/* 5) */
subsys_initcall(pcibios_init);
...
static void __init
pcibios_assign_resources(void){
  ...
}
```

6 Debug PCI

Symptomes :

- La carte du slot 1 ne boot pas.
- La 2ème carte ne mappe que les cartes présente sur son segment ; elle les mappent sur le 20.

```
$ grep -n 'CESRT_CPCI_SLOT21' $(find . -name '*')
./arch/powerpc/platforms/cesrio/pci.c:762:      rio_init_pci_slot(&cpci_slot_info[21], CESRT_CPCI_
./include/ces/cesResourceTable.h:207:  CESRT_CPCI_SLOT21
```

Nous allons chercher dans les directions donnees au paragraphe precedent traitant de l'initialisation de bus PCI.

6.1 Hack 1

Si l'on remplace la carte processeur par une autre on boot. En fait les adresses memoires lue pour la carte processeur sont mauvaises. En les remplacant ca marche :

- fichier *arch/powerpc/platforms/cesrio/pci.c* :

```
static void fixup_bibi(struct pci_dev *dev) {

  printk("RIO_PCI: fixup_bibi... \n");
  rio_print_pci_tree_dev(dev, 0,0);

  if (dev->resource[1].start == 0)
  {
    dev->resource[0].start = 0x42200000;
    dev->resource[0].end =  0x4220001f;
    dev->resource[0].flags = IORESOURCE_IO;
    dev->resource[1].start = 0x42300000;
    dev->resource[1].end =  0x4230001f;
    dev->resource[1].flags = IORESOURCE_MEM;
    printk("...RIO_PCI: Fixed up Bibi %x\n", dev->devfn);
  }
  rio_print_pci_tree_dev(dev, 0,0);
}
DECLARE_PCI_FIXUP_HEADER(0x10d9, 0x4065, fixup_bibi);
```

- fichier *drivers/pci/probe.c* :

```
/**
* pci_setup_device - fill in class and map information of a device
* @dev: the device structure to fill
*
* Initialize the device structure with information about the device's
* vendor,class,memory and IO-space addresses,IRQ lines etc.
* Called at initialisation of the PCI subsystem and by CardBus services.
* Returns 0 on success and -1 if unknown type of device (not normal, bridge
* or CardBus).
*/
static int pci_setup_device(struct pci_dev * dev)
{
  ...
  /* Early fixups, before probing the BARs */
  pci_fixup_device(pci_fixup_early, dev);

  switch (dev->hdr_type) {     /* header type */
    case PCI_HEADER_TYPE_NORMAL:     /* standard header */
    ...
    pci_read_bases(dev, 6, PCI_ROM_ADDRESS);
    ...
    break;
    ...
    /* We found a fine healthy device, go go go... */
    return 0;
  }

  static void pci_read_bases(struct pci_dev *dev, unsigned int howmany, int rom)
  {
    ...
    for(pos=0; pos<howmany; pos = next) {
```

```
        ...
        reg = PCI_BASE_ADDRESS_0 + (pos << 2);
pci_read_config_dword(dev, reg, &l);    <= ICI l=0 !!!
        ...
if ((l & PCI_BASE_ADDRESS_SPACE) ==
PCI_BASE_ADDRESS_SPACE_MEMORY) {
          ...
  res->start = l & PCI_BASE_ADDRESS_MEM_MASK;
  res->flags |= l & ~PCI_BASE_ADDRESS_MEM_MASK;
}
        ...
        res->start = l & PCI_BASE_ADDRESS_IO_MASK;
        res->flags |= l & ~PCI_BASE_ADDRESS_IO_MASK;
        ...
      }
    }


    void pci_device_add(struct pci_dev *dev, struct pci_bus *bus)
    {
      ...
      /* Fix up broken headers */
      pci_fixup_device(pci_fixup_header, dev);
      ...
    }

    /*
    * Read the config data for a PCI device, sanity-check it
    * and fill in the dev structure...
    */
    static struct pci_dev * __devinit
    pci_scan_device(struct pci_bus *bus, int devfn)
    {
      ...
      if (pci_setup_device(dev) < 0) {
        ...
      }

      struct pci_dev *pci_scan_single_device(struct pci_bus *bus, int devfn)
      {
        ...
        dev = pci_scan_device(bus, devfn);
        ...
        pci_device_add(dev, bus);
        ...
      }
```

Cela ne marche que sur le chassis 14 slots.

```
void
fixup_hess_pci_tree_dev(struct pci_dev *dev, u64 *rc)
{
  int i;
  struct resource *res;

  //printk(" %s %04x:%04x\n", pci_name(dev), dev->vendor, dev->device);

  // for each ressources
```

```
  for (i = 0; i < PCI_ROM_RESOURCE; i++) {
    res = &dev->resource[i];

    // we only look for MEM ressource (it works like that)
    if (res->flags & IORESOURCE_MEM) {
      //printk("  BAR %d: MEM 0x%08llx+0x%llx\n", i,
      //       (u64)res->start, (u64)(res->end - res->start + 1));
      if ((u64)res->end > *rc)
      *rc = res->end;
    }
  }
}

void
fixup_hess_pci_tree_bus(struct pci_bus *bus, const char *bus_name, u64 *rc)
{
  struct pci_dev *dev;

  //printk("%s BUS 0x%02x\n", bus_name, bus->number);

  /* Depth-First Search on bus tree */
  list_for_each_entry(dev, &bus->devices, bus_list) {
    if (dev->subordinate) {
      fixup_hess_pci_tree_bus(dev->subordinate, pci_name(dev), rc);
    } else {
      // looking for adress on the second or third segment only
      if (bus->number == 0x80 || bus->number == 0x88)
      fixup_hess_pci_tree_dev(dev, rc);
    }
  }
}

static void fixup_hess(struct pci_dev *dev)
{
  struct pci_bus *bus;
  struct pci_controller *hose;
  rio_pci_cfg_t *cfg;
  u64 rc = 0;

    if (dev->resource[0].start == 0)
    {
      printk("RIO_PCI: fixup_hess... \n");

      // adress on the second or third segment alway is biger or equal than :
      rc = 0x40000000 - 1;

      // for each bus
      list_for_each_entry(bus, &pci_root_buses, node) {
        // find the maximum MEM address allocated
        hose = bus->sysdata;
        cfg = hose->arch_data;
        fixup_hess_pci_tree_bus(bus, cfg->name, &rc);
      }
```

```
    dev->resource[0].start = rc + 1;
    dev->resource[0].end =   rc + 0x1f;
    dev->resource[0].flags = IORESOURCE_IO;
    dev->resource[1].start = rc + 0x100000;
    dev->resource[1].end =   rc + 0x10001f;
    dev->resource[1].flags = IORESOURCE_MEM;
    rio_print_pci_tree_dev(dev, 0,0);

    printk("...RIO_PCI: Fixed up Hess %x\n", dev->devfn);
  }
}
DECLARE_PCI_FIXUP_HEADER(0x10d9, 0x4065, fixup_hess);
```

6.2 Hack 2 : trouver le deuxieme bus depuis le second segment

Cela marche sur la carte RIO inseree dans le segment 2 et 3 d'un chassis 21 slots.
Il faudrait faire une option a passer au noyau
Le bridge 0x88 est vu comme etant le dev numero 1 sur le bus 0x20.
Fichier *arch/powerpc/platforms/cesrio/pci.c* :

```
int rio_pcibios_read_config(struct pci_bus *bus, unsigned int devfn, int offset,
 int len, u32 *val)
{
...
if (weAreNotOnFirstSegment &&
             bus->number == 0x20 && devfn == 8){
  pr_debug(":o) %s: sorry I enable brigge 80\n", __func__);
}
else
  if (hose->first_busno == bus->number &&
      !pci_slot_is_enabled(cfg, devno)) {
    *val = ~0;
    return PCIBIOS_DEVICE_NOT_FOUND;
...
}

int
rio_pcibios_write_config(struct pci_bus *bus, unsigned int devfn, int offset,
  int len, u32 val)
{
...
if (weAreNotOnFirstSegment &&
            bus->number == 0x20 && devfn == 8){
  pr_debug(":o) %s: sorry I enable brigge 80\n", __func__);
}
else
if (hose->first_busno == bus->number &&
    !pci_slot_is_enabled(cfg, devno)) {
return PCIBIOS_DEVICE_NOT_FOUND;
}
};
```

Par contre les drivers des cartes Hess ne marche losqu'ils emprunte le bus. On a la meme erreur avec le noyau original et la carte esclave directement sur le segment 3 :

```
/rMachine check in kernel mode.
Caused by (from SRR1=149030): Transfer error ack signal
Oops: Machine check, sig: 7 [#1]
```

Avec le noyau original sur la carte master : Slave sur le segment 1 : atteint tous les segments
Slave sur le segment 2 : atteint les segments 2 (ssi la carte rio est a droite des autres) et 3
Slave sur le segment 3 : n'atteint pas le segment 3

Avec le noyau modifie par le hack 3 : Slave sur le segment 1 : atteint tous les segments
Slave sur le segment 2 : atteint le segments 2
Slave sur le segment 3 : n'atteint pas le segment 3

6.3 Hack 3 : ignorer la carte rio exclave lors du boot de la carte maitre

A tester :
Slave sur le segment 1 : ok
Slave sur le segment 2 :
Slave sur le segment 3 :
Fichier *drivers/pci/probe.c* :

```
/*
 * Read the config data for a PCI device, sanity-check it
 * and fill in the dev structure...
 */
static struct pci_dev * __devinit
pci_scan_device(struct pci_bus *bus, int devfn)
{
...
/* some broken boards return 0 or ~0 if a slot is empty: */
if (l == 0xffffffff || l == 0x00000000 ||
    l == 0x0000ffff || l == 0xffff0000 ||
    l == 0x406510d9) /* id de la rio slave */
return NULL;
...
}
```

6.4 Patch : hack 2 et 3

Fichier *arch/powerpc/platforms/cesrio/pci.c* :

```
/***** Add for Hess2 *****/

static int slaveInsertedOnSecondSegment = 0;
static int __init parse_slaveInsertedOnSecondSegment(char *arg)
{
        slaveInsertedOnSecondSegment = 1;
        return 0;
}
early_param("seg_2", parse_slaveInsertedOnSecondSegment);

static void fixup_hess2_enable2ndBridge(struct pci_dev *dev) {
  if (slaveInsertedOnSecondSegment){
    printk("RIO_PCI: %s call by slave rio on 21 slot crate\n", __func__);

    pci_slot_info[2][1].visible=1;      // enable device
  }
}
DECLARE_PCI_FIXUP_EARLY(0x8086, 0x1008, fixup_hess2_enable2ndBridge);

static void fixup_hess2_dontMapSlave(struct pci_dev *dev) {
  printk("RIO_PCI: %s call by master rio on 21 slot crate\n", __func__);

  dev->class = PCI_CLASS_NOT_DEFINED;  // don't map device
  dev->hdr_type = 3;                   // free device
}
DECLARE_PCI_FIXUP_EARLY(0x10d9, 0x4065, fixup_hess2_dontMapSlave);

/* end Add for Hess2 *****/
```

6.5 Piste 3

On utilise necessairement l'option `cpci_full_enum` pour remapper les cartes. Il s'agit de regarder ce que fait cette option qui a ete ajoute recement par CES pour permettre le boot de plusieurs cartes processeurs. L'idee serait de la modifier pour que le boot sur les different segments donnemt les memes adresses aux ressources des cartes PCI d'une rio a l'autre.

- fichier *arch/powerpc/platforms/cesrio/pci.c* :

```
static void __init rio_pcibios_fixup(void) {
  ...
  rio_pci_allocate_bus_resources(bus, io_res, mem_res);
  ...
}

void __init rio_pci_allocate_bus_resources( ...
list_for_each_entry(bus, &parent_bus->children, node) {
  rio_pci_allocate_bus_resources(bus, io_res, mem_res);
  update_bridge_resource(bus->self, io_res);
  update_bridge_resource(bus->self, mem_res);
  list_for_each_entry(dev, &parent_bus->devices, bus_list) {
    pci_update_resource(dev, res, i);
  }
```

- fichier *arch/powerpc/kernel/pci_32.c* :

```
  void __init
  update_bridge_resource(struct pci_dev *dev, struct resource *res)
  {
    pci_read_config_word(dev, PCI_COMMAND, &cmd);
    ...
  }

  void pcibios_resource_to_bus(struct pci_dev *dev, struct pci_bus_region *region,
  struct resource *res)
  {
    ...
  }
  EXPORT_SYMBOL(pcibios_resource_to_bus);
```

- fichier */include/linux/pci.h* :

```
  static inline int pci_read_config_word(struct pci_dev *dev, int where, u16 *val)
  {
    return pci_bus_read_config_word (dev->bus, dev->devfn, where, val);
  }
```

- fichier */driver/pci/access.c* :

```
  #define PCI_OP_READ(size,type,len) \
  int pci_bus_read_config_##size \
  (struct pci_bus *bus, unsigned int devfn, int pos, type *value) \
  {...

    EXPORT_SYMBOL(pci_bus_read_config_word);
```

- fichier *drivers/pci/setup-res.c* :

```
void
pci_update_resource(struct pci_dev *dev, struct resource *res, int resno)
{
  ...
  pcibios_resource_to_bus(dev, &region, res);
  ...
  pci_write_config_dword(dev, reg, new);
  ...
}
```

6.6 Piste 4

Il s'agit de la dernière erreur relevé dans le cas défaillant sur la carte du slot 1. A priori l'erreur est due au fait que la carte est mappee a 5M alors que la valeur max du bus est en dessous.

- fichier *arch/powerpc/kernel/pci_32.c* : fichier appelant

```
static int __init
pcibios_init(void)
{
  ...
  // un seul appel pour le bus 20
  pcibios_assign_resources();
  ...
}
subsys_initcall(pcibios_init);

static void __init
pcibios_assign_resources(void)
{
  ...
  printk(":@) arch/powerpc/kernel/pcibios_assign_resources 1\n");
  rc = pci_assign_resource(dev, idx);
  BUG_ON(rc);
  ...
}
```

- fichier *drivers/pci/setup-res.c* : `PCI: Failed to allocate mem ressource #0:10000000@50000000 for 0001:8`

```
int pci_assign_resource(struct pci_dev *dev, int resno)
{
  ...
  size = res->end - res->start + 1; // bus->number=20 size=268435456 !!
  ...
  ret = pci_bus_alloc_resource(bus, res, size, align, min, ...
  if (ret) {
    printk(KERN_ERR ":@/ PCI: Failed to allocate %s resource "
    ...
```

- fichier *drivers/pci/bus.c* :

```
int
pci_bus_alloc_resource(struct pci_bus *bus, struct resource *res, ...
{
  ret = allocate_resource(r, res, size,.. // res[1]
}
```

- fichier *kernel/resource.c* :

```
/**
 * allocate_resource - allocate empty slot in the resource tree given range & alignment
 * @root: root resource descriptor
 * @new: resource descriptor desired by caller
 * @size: requested resource region size
 * @min: minimum size to allocate
 * @max: maximum size to allocate
 * @align: alignment requested, in bytes
 * @alignf: alignment function, optional, called if not NULL
 * @alignf_data: arbitrary data to pass to the @alignf function
```

```
 */
int allocate_resource(struct resource *root, struct resource *new,
      resource_size_t size, resource_size_t min,
      resource_size_t max, resource_size_t align,
      void (*alignf)(void *, struct resource *,
     resource_size_t, resource_size_t),
      void *alignf_data)
{
...
err = find_resource(root, new, size, min, max, align, alignf, alignf_data);
...
return err;
}

/*
 * Find empty slot in the resource tree given range and alignment.
 */
static int find_resource(struct resource *root, struct resource *new,
 resource_size_t size, resource_size_t min,
 resource_size_t max, resource_size_t align,
 void (*alignf)(void *, struct resource *,
resource_size_t, resource_size_t),
 void *alignf_data)
{
struct resource *this = root->child;

new->start = root->start;
/*
 * Skip past an allocated resource that starts at 0, since the assignment
 * of this->start - 1 to new->end below would cause an underflow.
 */
if (this && this->start == 0) {
new->start = this->end + 1;
this = this->sibling;
}
for(;;) {
if (this)
new->end = this->start - 1;
else
new->end = root->end;
if (new->start < min)
new->start = min;
if (new->end > max)
new->end = max;
new->start = ALIGN(new->start, align);
if (alignf)
alignf(alignf_data, new, size, align);
if (new->start < new->end && new->end - new->start >= size - 1) {
new->end = new->start + size - 1;
return 0;
}
if (!this)
break;
new->start = this->end + 1;
this = this->sibling;
}
printk("ARFFFF\n");
return -EBUSY;
```

```
}
```