New hotplug handling method for UPCIEDEV driver

Advantages of new hotplug handling method

Davit Kalantaryan
Zeuthen, 05.07.2021

List of advantages

> CPU efficient

> Memory efficient

> More information on hotplug

> Less code with less complicated dependency and therefore easy maintenance

> Possibility for further improvement of driver

CPU efficient

> During device remove callback call no iteration thru files

> No unnecessary device switch to the null device in the device remove callback

Memory efficient

> No need for the list of files for each board

> No need for the several fields of the structure pciedev_dev

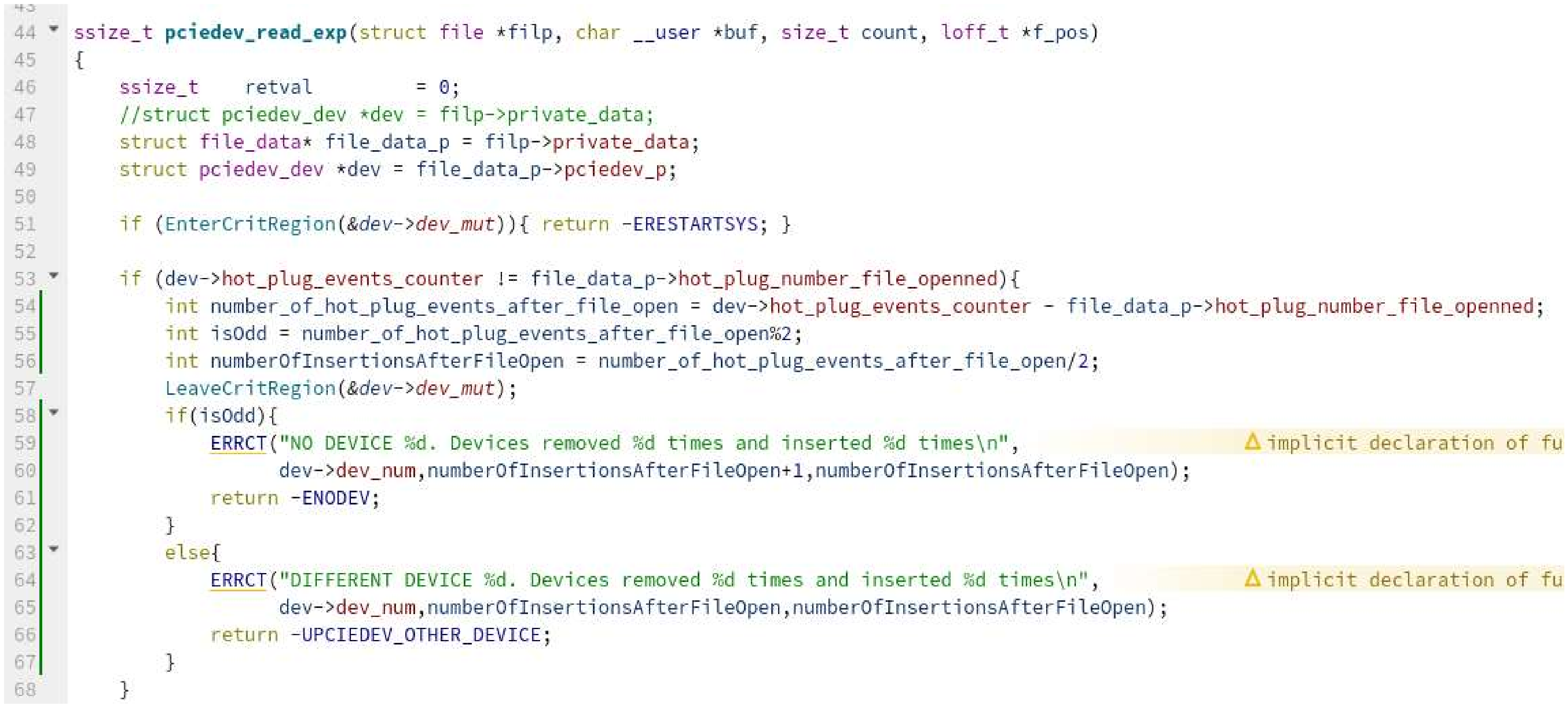
- Field ‘int binded;’
- Fileld ‘int dev_sts;’
- Field ‘struct pciedev_cdev *parent_dev;’
- Field ‘struct upciedev_file_list dev_file_list;’

> No need for the field ‘pciedev_dev *pciedev_dev_m[PCIEDEV_NR_DEVS + 1];’ for the structure ‘pciedev_cdev’.

> Another dramatic memory usage improvement will be described in slide 7

More information on hotplug

- With the old approach in the case of error, we only know that the device was removed.
- With the new approach, kernel-space driver has more information and this information can be delivered to user space application by means of return or simple logs can be done for host admin. Just consider the code snippet below



```
ssize_t pciedev_read_exp(struct file *filp, char __user *buf, size_t count, loff_t *f_pos)
{
    ssize_t    retval          = 0;
    //struct pciedev_dev *dev = filp->private_data;
    struct file_data* file_data_p = filp->private_data;
    struct pciedev_dev *dev = file_data_p->pciedev_p;

    if (EnterCritRegion(&dev->dev_mut)){ return -ERESTARTSYS; }

    if (dev->hot_plug_events_counter != file_data_p->hot_plug_number_file_openned){
        int number_of_hot_plug_events_after_file_open = dev->hot_plug_events_counter - file_data_p->hot_plug_number_file_openned;
        int isOdd = number_of_hot_plug_events_after_file_open%2;
        int numberOfInsertionsAfterFileOpen = number_of_hot_plug_events_after_file_open/2;
        LeaveCritRegion(&dev->dev_mut);
        if(isOdd){
            ERRCT("NO DEVICE %d. Devices removed %d times and inserted %d times\n",          ⚠implicit declaration of fu
                  dev->dev_num,numberOfInsertionsAfterFileOpen+1,numberOfInsertionsAfterFileOpen);
            return -ENODEV;
        }
        else{
            ERRCT("DIFFERENT DEVICE %d. Devices removed %d times and inserted %d times\n",   ⚠implicit declaration of fu
                  dev->dev_num,numberOfInsertionsAfterFileOpen,numberOfInsertionsAfterFileOpen);
            return -UPCIEDEV_OTHER_DEVICE;
        }
    }
```

Less code and therefor easy maintenance

- No code concerning device files list handling
- No complicated dependent code necessary for getting pointer of the null device from the device remove callback.

Possibility for further improvement of driver

- Currently, with each specific device driver, 16 pciedev_dev structures are allocated. This code design comes from fact that each pciedev_dev structure needs a pointer to the device null and this means a pointer to pciedev_cdev structure for the specific driver that in turn allocates 16 pciedev_dev (too complicated ☺).
  Imagine that we have 16 slot mTCA and this means possible 14 different devices with specific drivers using upciedev driver as a parent, then we will have 16x14=224 pciedev_dev structures allocated.
  Generally speaking for each host maximum needed count of pciedev_dev structures can be equal to the number of slots for the cluster handled by the CPU.
- !!! This modification should not take place immediately, because this can break compilation or even worse runtime behavior of dependent drivers.
- !!! Important: Even if this modification (not creating 16 pciedev_dev for each specific driver) should wait for careful tests for all dependent drivers, modification of getting rid of callback dependence on device null (that’s why on driver global structure) should be done soon (immediately) in order to make a chance for new (also some old) drivers to implement optimal code without using huge structure pciedev_cdev.