Linux Device Drivers - char driver

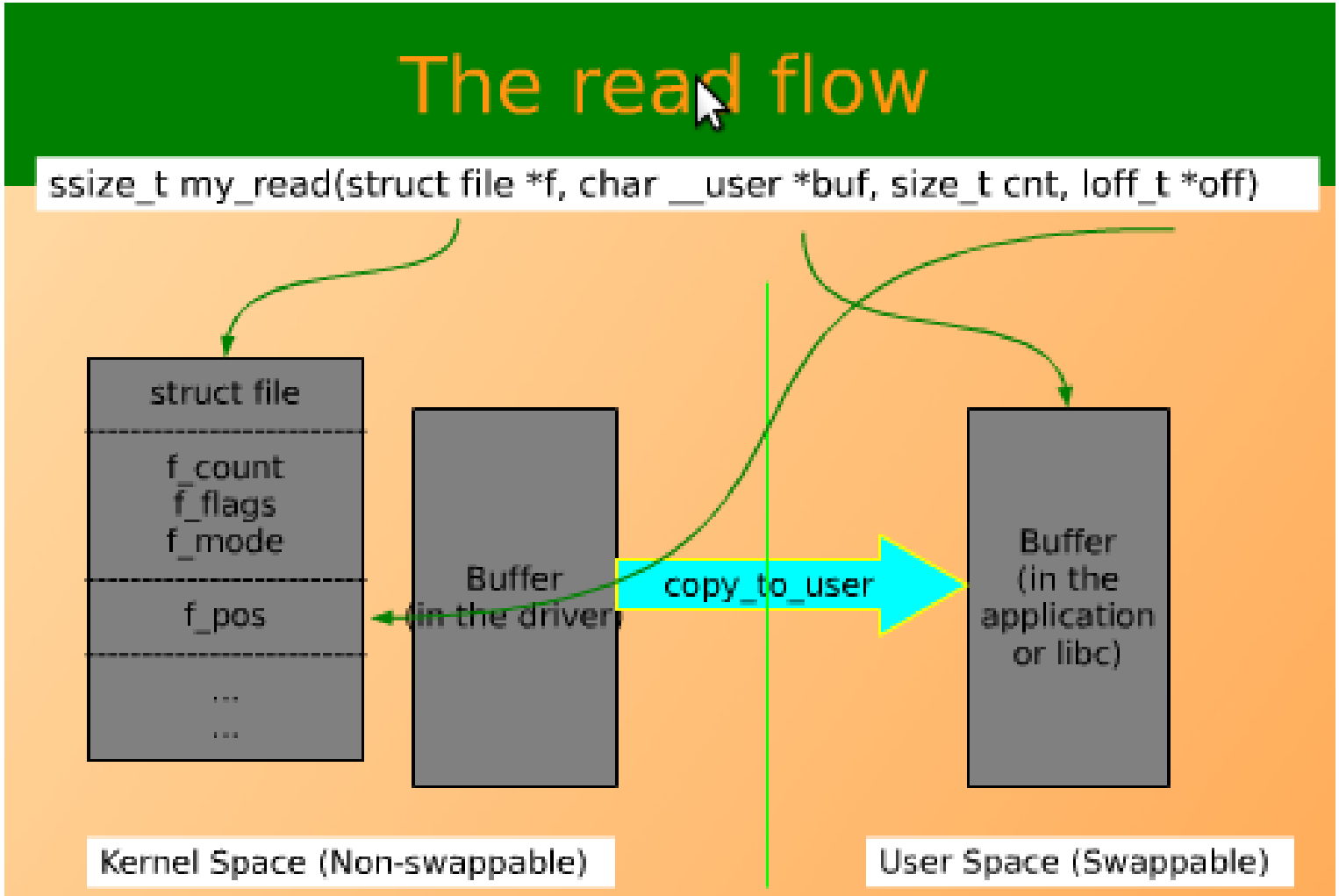
Jernej Vičič

Overview

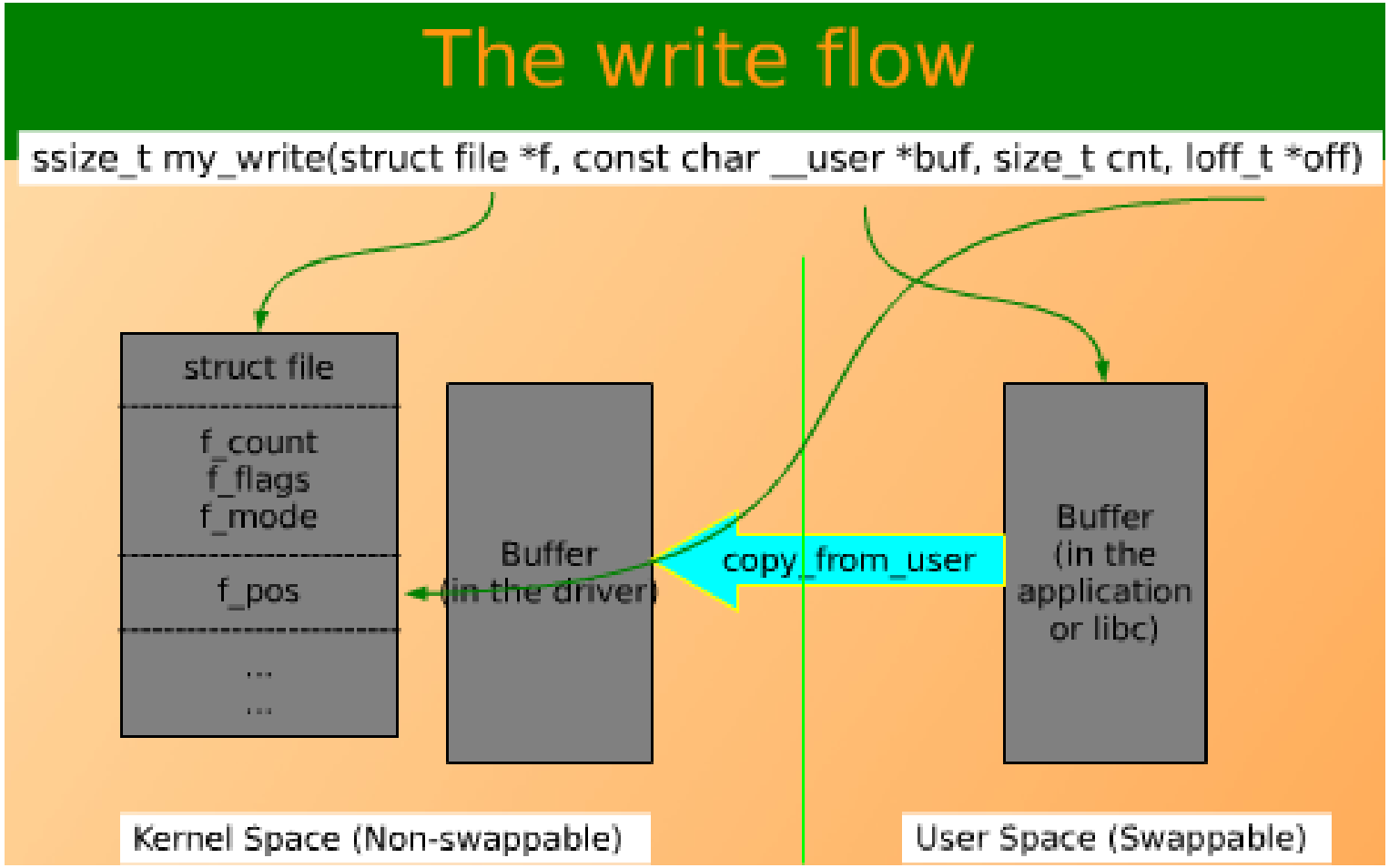
Introduction

- simplest driver,
- suitable for most simple devices,
- follow the book.

Reading


The read flow
ssize_t my_read(struct file *f, char __user *buf, size_t cnt, loff_t *off)
struct file
f_count
f_flags
f_mode
f_pos
...
...
Buffer
(in the driver)
copy_to_user
Buffer
(in the
application
or libc)
Kernel Space (Non-swappable)
User Space (Swappable)

Writing


The write flow
ssize_t my_write(struct file *f, const char __user *buf, size_t cnt, loff_t *off)
struct file
f_count
f_flags
f_mode
f_pos
...
...
Buffer
(in the driver)
copy_from_user
Buffer
(in the
application
or libc)
Kernel Space (Non-swappable)
User Space (Swappable)

scull

- *scull* (Simple Character Utility for Loading Localities),
- character driver,
- uses part of the memory to simulate the device,
- device does not do *anything* useful,
- suitable only for demonstration,
- shows the interfaces between the kernel and the character driver,
- suitable as a template for your drivers,
- shows the actual installation and startup.

SCULL architecture

- define capabilities,
- multiple types of devices.

Devices

- part of memory, global, persistent (keeps the values);
- user can use cp, cat, ...
- scull0,
- scull1,
- scull2,
- scull3.

Devices, continue

- 4 FIFO devices,
- one process reads what the other wrote,
- scullpipe0,
- scullpipe1,
- scullpipe2,
- scullpipe3.

Devices, continue

- special version of memory buffer (similar to scull0);
- scullsingle, only one process at once;
- scullpriv, private for every console (tty);
- sculluid, only one user can access - device busy;
- scullwuid, only one use can access - blocking open.

Major, minor

- user can access these devices through file system;
- */dev*;
- observe the *c* tag;
- in *ls -l*:

Device overview

```
crw-rw-rw- 1 root root 1, 3 Apr 11 2002 null
crw------- 1 root root 10, 1 Apr 11 2002 psaux
crw------- 1 root root 4, 1 Oct 28 03:04 tty1
crw-rw-rw- 1 root tty 4, 64 Apr 11 2002 ttys0
crw-rw---- 1 root uucp 4, 65 Apr 11 2002 ttyS1
crw--w---- 1 vcsa tty 7, 1 Apr 11 2002 vcs1
crw--w---- 1 vcsa tty 7, 129 Apr 11 2002 vcsa1
crw-rw-rw- 1 root root 1, 5 Apr 11 2002 zero
```

Explanation

- major numbers: 1, 4, 7, and 10;
- minor numbers: 1, 3, 5, 64, 65, 129;
- example: /dev/null and /dev/zero are controlled by the same driver (1);

Inner representation of device numbers

- type *dev_t*, defined in <linux/types.h>;
- represents major and minor numbers;
- 32 bits:
- 12 bits fo major, 20 bits for minor;
- available macros for translation into *dev_t*:

```
MAJOR(dev_t dev);
MINOR(dev_t dev);
MKDEV(int major, int minor);
```

Allocation and release of device numbers

```
int register_chrdev_region(dev_t first, unsigned int count, char *name)
```

- defined in *<linux/fs.h>*;
- *first*, start of region;
- *count*, number of requested numbers;
- *name*, name of device, shown in */proc/devices*

Allocation and release of device numbers

```
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,
    unsigned int count, char *name);
```

- *dev*, allocate major number;
- *firstminor*, the smallest minor number;
- *count*, the number of numbers we request;
- *name*, the device name, will appear in */proc/devices*

Allocation and release of device numbers

```
void unregister_chrdev_region(dev_t first, unsigned int count);
```

- release number region;

Dynamic number allocation

```
void unregister_chrdev_region(dev_t first, unsigned int count);
```

- better use *alloc_chrdev_region*;
- script:

Dynamic number allocation

```
#!/bin/sh
module="scull"
device="scull"
mode="664"
# invoke insmod with all arguments we got
# and use a pathname, as newer modutils don't look in . by default
/sbin/insmod ./$module.ko $* || exit 1
# remove stale nodes
rm -f /dev/${device}[0-3]
major=$(awk "\\$2= =\"$module\" {print \\$1}" /proc/devices)
mknod /dev/${device}0 c $major 0
mknod /dev/${device}1 c $major 1
mknod /dev/${device}2 c $major 2
mknod /dev/${device}3 c $major 3
# give appropriate group/permissions, and change the group.
# Not all distributions have staff, some have "wheel" instead.
group="staff"
grep -q '^staff:' /etc/group || group="wheel"
chgrp $group /dev/${device}[0-3]
chmod $mode /dev/${device}[0-3]
```

Data structures

- File Operations
- The file Structure
- The inode Structure

File Operations

- *file_operations*,
- defined in <*linux/fs.h*>,
- usualy named *fops*,
- each field points to a function or to NULL,

File Operations - list

- struct module *owner
- loff_t (*llseek) (struct file *, loff_t, int);
- ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
- ssize_t (*aio_read)(struct kiocb *, char __user *, size_t, loff_t);
- ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
- ssize_t (*aio_write)(struct kiocb *, const char __user *, size_t, loff_t *);
- int (*readdir) (struct file *, void *, filldir_t);

File Operations - list continue

- unsigned int (*poll) (struct file *, struct poll_table_struct *);
- int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
- int (*mmap) (struct file *, struct vm_area_struct *);
- int (*open) (struct inode *, struct file *);
- int (*flush) (struct file *);
- int (*release) (struct inode *, struct file *);
- int (*fsync) (struct file *, struct dentry *, int);
- ...

File Operations - scull

```
struct file_operations scull_fops = {
   .owner = THIS_MODULE,
   .llseek = scull_llseek,
   .read = scull_read,
   .write = scull_write,
   .ioctl = scull_ioctl,
   .open = scull_open,
   .release = scull_release,
};
```

The file Structure

- *struct file*,
- defined in *<linux/fs.h>*,
- no connection with FILE in userspace programs,
- represents an open file,

File Operations - list

- mode_t f_mode;
- loff_t f_pos;
- unsigned int f_flags;
- struct file_operations *f_op;
- void *private_data;
- struct dentry *f_dentry;

The inode structure

- kernel uses to represent files,
- two important fields:
- *dev_t i_rdev* major number for inodes that represent devices ,
- *struct cdev *i_cdev*, character devices

The inode structure - macros

```
unsigned int iminor(struct inode *inode);
unsigned int imajor(struct inode *inode);
```

Register char device

- char devices are in presented to the kernel with *struct cdev*;
- defined in v *<linux/cdev.h>*;
- two ways of initialization: independent structure, part of private structures

Independent structure

```
struct cdev *my_cdev = cdev_alloc( );
my_cdev->ops = &my_fops;
instruct the kernel:
int cdev_add(struct cdev *dev, dev_t num, unsigned int count);
```

Part of private structures

- used by scull;
- use next initialization;

```
void cdev_init(struct cdev *cdev, struct file_operations *fops);
```

- povemo jedru;

```
int cdev_add(struct cdev *dev, dev_t num, unsigned int count);
```

Delete the device

```
void cdev_del(struct cdev *dev);
```

Structure

```
struct scull_dev {
  struct scull_qset *data; /* Pointer to first quantum set */
  int quantum; /* the current quantum size */
  int qset; /* the current array size */
  unsigned long size; /* amount of data stored here */
  unsigned int access_key; /* used by sculluid and scullpriv */
  struct semaphore sem; /* mutual exclusion semaphore */
  struct cdev cdev; /* Char device structure */
};
```

Initialization and structure add

```
static void scull_setup_cdev(struct scull_dev *dev, int index)
{
  int err, devno = MKDEV(scull_major, scull_minor + index);
  cdev_init(&dev->cdev, &scull_fops);
  dev->cdev.owner = THIS_MODULE;
  dev->cdev.ops = &scull_fops;
  err = cdev_add (&dev->cdev, devno, 1);
  /* Fail gracefully if need be */
  if (err)
    printk(KERN_NOTICE "Error %d adding scull%d", err, index);
}
```

Method open

- ussually take care of:
  - check for error in the device (device-not-ready, ...);
  - initialize the device if first open;
  - change the *f_op* pointer (if needed)
  - alocate and fille the for *filp->private_data*

Our example

```
use macro container_of(pointer, container_type, container_field);

struct scull_dev *dev; /* device information */
dev = container_of(inode->i_cdev, struct scull_dev, cdev);
filp->private_data = dev; /* for other methods */
```

Our example, continue

```
int scull_open(struct inode *inode, struct file *filp)
{
  struct scull_dev *dev; /* device information */
  dev = container_of(inode->i_cdev, struct scull_dev, cdev);
  filp->private_data = dev; /* for other methods */
  /* now trim to 0 the length of the device if open was write-only */
  if ( (filp->f_flags & O_ACCMODE) = = O_WRONLY) {
    scull_trim(dev); /* ignore errors */
  }
  return 0; /* success */
}
```

The release method

- dealocation of everything open methoda allocated in $filp->private\_data$;
- closes the device at last close.

Our example

```
int scull_release(struct inode *inode, struct file *filp)
{
  return 0;
}
```

Memory

- How do we allocate memory?
- Example scull.

Memory allocation

```
<linux/slab.h>

void *kmalloc(size_t size, int flags);
void kfree(void *ptr);
```

- *flags*: simplest/most used GFP_KERNEL

Parts of memory (quantum)

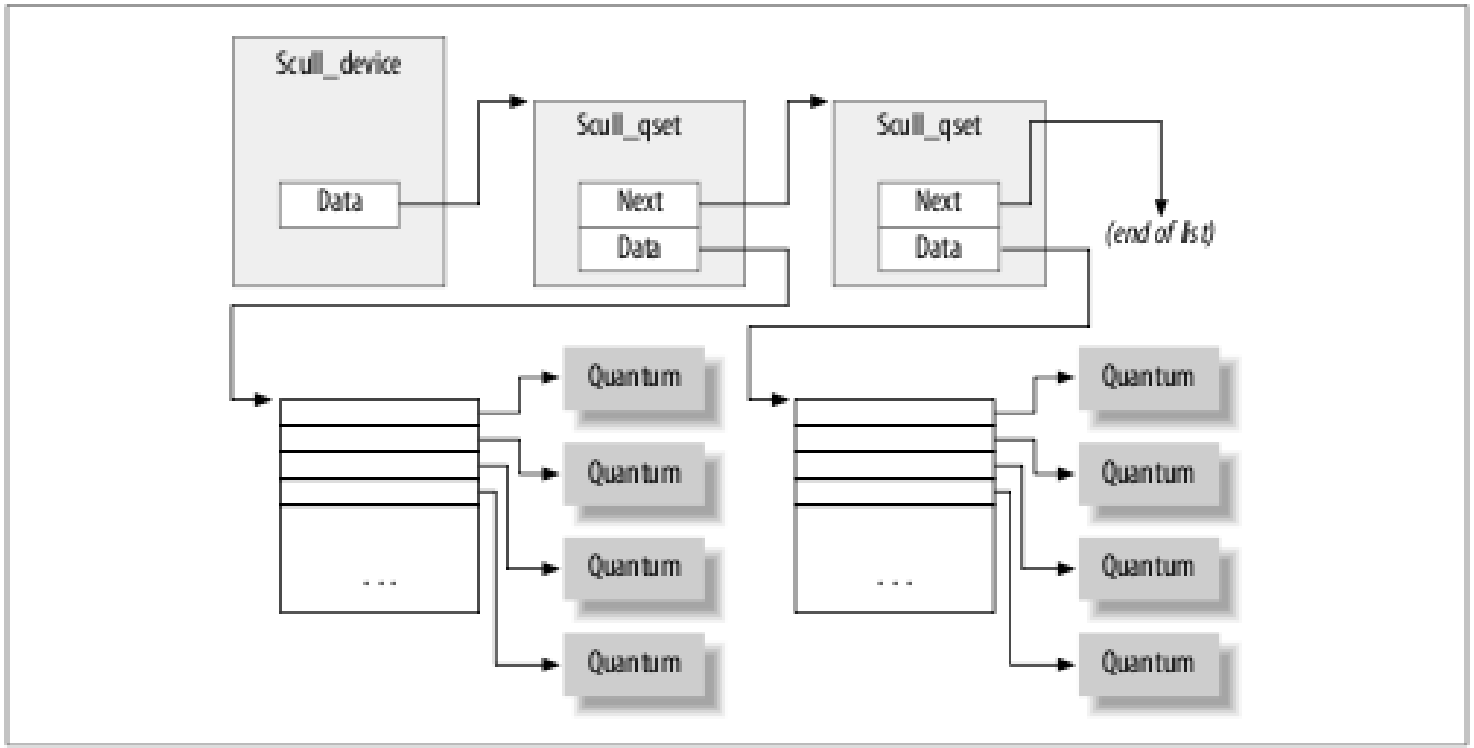


Figure: Memory quantum.

Memory quantum

```
struct scull_qset {
  void **data;
  struct scull_qset *next;
};
```

Memory quantum

```
int scull_trim(struct scull_dev *dev)
{
  struct scull_qset *next, *dptr;
  int qset = dev->qset; /* "dev" is not-null */
  int i;
  for (dptr = dev->data; dptr; dptr = next) { /* all the list items */
    if (dptr->data) {
      for (i = 0; i < qset; i++)
        kfree(dptr->data[i]);
      kfree(dptr->data);
      dptr->data = NULL;
    }
    next = dptr->next;
    kfree(dptr);
  }
  dev->size = 0;
  dev->quantum = scull_quantum;
```

Read and write

```
ssize_t read(struct file *filp, char __user *buff, size_t count, loff_t
ssize_t write(struct file *filp, const char __user *buff, size_t count,
```

- *filp* file pointer;
- *count* data size;
- *buff* pointer to data;
- *offp* pointer to "long offset type" object that defines file position;
- *buff* is in user space

Copy into and from user space

```
unsigned long copy_to_user(void __user *to,
const void *from,
unsigned long count);

unsigned long copy_from_user(void *to,
const void __user *from,
unsigned long count);
```

Read implementation

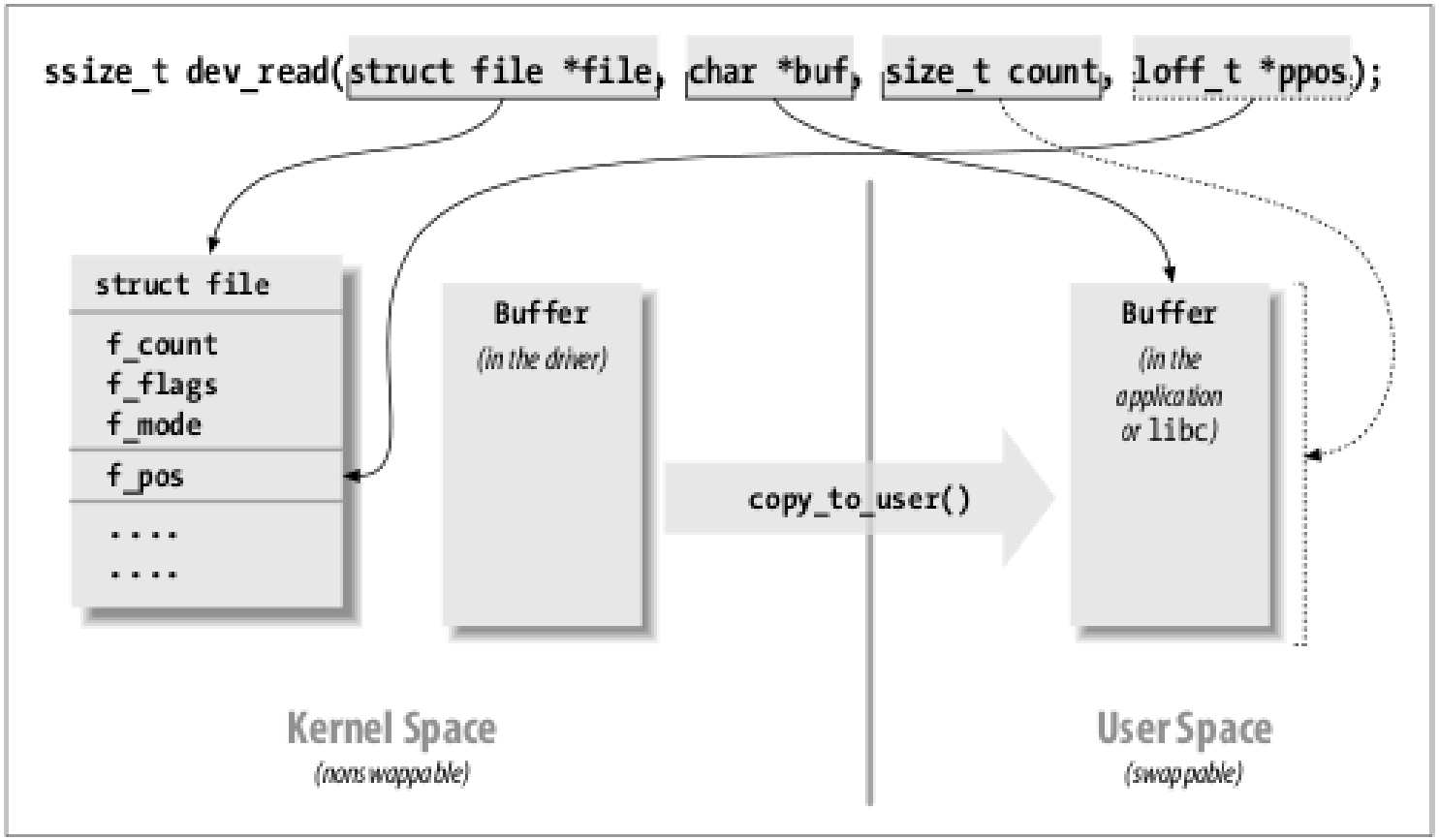


Figure: Implementacija branja.

```
ssize_t scull_read(struct file *filp, char __user *buf, size_t count,
loff_t *f_pos)
{
  struct scull_dev *dev = filp->private_data;
  struct scull_qset *dptr; /* the first listitem */
  int quantum = dev->quantum, qset = dev->qset;
  int itemsize = quantum * qset; /* how many bytes in the listitem */
  int item, s_pos, q_pos, rest;
  ssize_t retval = 0;
  if (down_interruptible(&dev->sem))
    return -ERESTARTSYS;
  if (*f_pos >= dev->size)
    goto out;
  if (*f_pos + count > dev->size)
    count = dev->size - *f_pos;
```

```
  /* find listitem, qset index, and offset in the quantum */
  item = (long)*f_pos / itemsize;
  rest = (long)*f_pos % itemsize;
  s_pos = rest / quantum; q_pos = rest % quantum;
  /* follow the list up to the right position (defined elsewhere) */
  dptr = scull_follow(dev, item);
  if (dptr = = NULL || !dptr->data || ! dptr->data[s_pos])
    goto out; /* don't fill holes */
  /* read only up to the end of this quantum */
  if (count > quantum - q_pos)
    count = quantum - q_pos;
  if (copy_to_user(buf, dptr->data[s_pos] + q_pos, count)) {
    retval = -EFAULT;
    goto out;
  }
  *f_pos += count;
  retval = count;
  out:
  up(&dev->sem);
  return retval;
}
```

```
ssize_t scull_write(struct file *filp, const char __user *buf, size_t count,
loff_t *f_pos)
{
  struct scull_dev *dev = filp->private_data;
  struct scull_qset *dptr;
  int quantum = dev->quantum, qset = dev->qset;
  int itemsize = quantum * qset;
  int item, s_pos, q_pos, rest;
  ssize_t retval = -ENOMEM; /* value used in "goto out" statements */
  if (down_interruptible(&dev->sem))
    return -ERESTARTSYS;
  /* find listitem, qset index and offset in the quantum */
  item = (long)*f_pos / itemsize;
  rest = (long)*f_pos % itemsize;
  s_pos = rest / quantum; q_pos = rest % quantum;
  /* follow the list up to the right position */
  dptr = scull_follow(dev, item);
  if (dptr = = NULL)
    goto out;
```

```
if (!dptr->data) {
  dptr->data = kmalloc(qset * sizeof(char *), GFP_KERNEL);
  if (!dptr->data)
  goto out;
  memset(dptr->data, 0, qset * sizeof(char *));
}
if (!dptr->data[s_pos]) {
  dptr->data[s_pos] = kmalloc(quantum, GFP_KERNEL);
  if (!dptr->data[s_pos])
    goto out;
}
/* write only up to the end of this quantum */
if (count > quantum - q_pos)
  count = quantum - q_pos;
if (copy_from_user(dptr->data[s_pos]+q_pos, buf, count)) {
  retval = -EFAULT;
  goto out;
}
*f_pos += count;
retval = count;
/* update the size */
if (dev->size < *f_pos)
```