CONCEPTES AVANÇATS DE SISTEMES OPERATIUS (CASO)

Facultat d'Informàtica de Barcelona, Dept. d'Arquitectura de Computadors, curs 2023/2024 – 2Q

Pràctiques de laboratori Dispositius de bloc

Material

La vostra instal·lació de Linux - aquesta pràctica no la tenim per MacOSX.

En el sistema de treball, els headers de linux instal·lats a /usr/src/linux-headers-<versio-del-kernel>
i apuntats correctament des de </lib/modules/versio-del-kernel>/build.

Implementació de dispositius de tipus bloc

El funcionament dels dispositius de bloc té semblances amb el dels de caràcter, però també té diferències.

Tot i que els dispositius de bloc també s'han d'enregistrar (amb la funció *register_blkdev*), les operacions equivalents als dispositius de caràcter s'usen ben poc. Només per obrir, tancar i fer alguna operació de consulta de les característiques del dispositiu, com ara la geometria.

En aquesta pràctica completareu el desenvolupament d'un dispositiu de bloc que farà la funció d'un *ramdisk*. Podreu utilitzar un esquema de mòdul amb dispositiu de bloc ja escrit i el completareu amb les següents funcions:

- Agafar i alliberar la memòria que representarà el vostre disc en RAM.
- Implementar l'operació que, donat el dispositiu, retorna la seva geometria (xrd_getgeo).
- Implementar parts de l'operació que atèn a les peticions de lectura i escriptura (a través de xrd_submit_bio i xrd_rw_page). En particular, implementar les funcions copy_from_xrd i copy_to_xrd.

Per a fer-ho, seguiu les següents seccions pas a pas.

Els fitxers de suport

Al costat d'aquesta pràctica trobareu aquests fitxers, mireu-los bé, que s'aniran explicant al llarg de l'enunciat:

Makefile - fitxer per compilar el mòdul myblkdrv.ko

interface.h - les estructures de dades i definicions necessàries

interface.c - la interfície del mòdul (init/fini) i les funcions necessàries per fer funcionar el disc

implementation.c - fitxer amb les funcions que heu d'implementar vosaltres

check-disk.sh, check-partition.sh - shell scripts que permeten verificar que el disc funciona correctament

ATENCIÓ: els fitxers proporcionats (casoblk-5.15.tar.bz2**) compilen correctament per a la versió 5.15.xx del kernel de Linux. Si teniu una instal·lació del kernel diferent, comenteu-ho amb el professor, per obtenir el codi apropiat a la**

vostra versió.

La representació del *ramdisk*

Dins el kernel tindrem una estructura amb aquesta informació per representar el nostre *ramdisk*:

```
// Estructura que defineix el nostre disc
struct xrd_device {
   int                    xrd_number;          // numero del disc, 0, 1...
   struct gendisk       * xrd_disk;            // punter a un disc generic
   struct list_head       xrd_list;            // lista de discos d'aquest tipus
   char                 * disk_memory;         // punter a les dades del disc
   u64                    size;      // mida total en bytes de les dades del disc
};
```

Aquesta estructura ja està definida al fitxer interface.h, i s'omple en la funció `xrd_alloc` (interface.c).

Agafar i alliberar la memòria del disc

Per demanar la memòria que haurà de mantenir les dades del disc, implementarem la funció:

```
char * alloc_disk_memory(unsigned long size);
```

Per a fer-ho, **editeu** el fitxer **implementation.c** i observeu que heu de situar la vostra implementació entre els comentaris:

```
      // la vostra implementació va aquí



     // fi de la vostra implementació
```

Aquesta funció ha d'usar el servei intern del kernel **vmalloc(size)** per demanar la memòria de dades del disc. Aquesta memòria estarà en l'espai del kernel. La interfície de vmalloc és (podeu veure-ho a `linux/vmalloc.h`):

```
void *vmalloc(unsigned long size);
```

Per a fer-ho, useu:

```
char * p = vmalloc(PAGE_ALIGN(size));
```

PAGE_ALIGN és una macro que torna la mida superior a la demanada que és múltiple de pàgina. En el nostre cas, no faria falta fer-la servir, perquè el disc té mida 16MB, però així assegurem que si canviem de mida, no ens hem de preocupar.

Inicialitzeu aquesta memòria de principi a fi, dins la vostra funció `alloc_disk_memory`, amb un patró conegut, de forma que al llegir del disc, sense haver-hi escrit, obtinguem aquesta informació i ens permeti veure que les lectures del disc funcionen bé (veure secció "Proves de lectura del disc").

La memòria del disc s'agafarà en insertar el mòdul myblkdrv.ko. Si a l'eliminar el mòdul del sistema no alliberem la memòria del disc, haurem introduït un *memory leak*. Per evitar-lo, abans d'insertar el mòdul, programeu també la rutina:

```
void free_disk_memory (char * disk_mem);
```

Aquesta funció haurà d'usar:

```
void vfree(const void *addr);
```

per alliberar la memòria agafada abans.

Si durant el desenvolupament us interessa que les funcions de suport donin més informació al dmesg, podeu activar la macro -DDEBUG al compilar o afegir #define DEBUG 1 al fitxer interface.h.

Proves inicials amb el disc

Un cop tingueu implementades les rutines per demanar i alliberar la memòria pel *ramdisk*, feu les següents proves. Per aquelles proves que incloguin lectures i escriptures, si voleu veure com haurien de funcionar en un *ramdisk* de Linux, podeu usar el dispositiu ja existent[1] `/dev/ram0`.

1) Carregueu el mòdul amb `insmod` i feu un `dmesg`. Si tot va bé veureu missatges d'aquest estil:

```
[412178.767384] Hello All World!
[412178.767384] alloc_disk_memory 16777216 address 0000000000000000
[412178.767384] init returns -12
[415631.938189] Hello world! max_part 3 part_shift 2 range 8
[415631.938189] alloc_disk_memory 16777216 address 0000000074f264cf
[415631.938189] alloc_disk_memory 16777216 address 0000000074f264cf
[415631.938189] xrd_alloc returns 000000001872b8b5 (0000000074f264cf)
[415631.938189] After xrd_alloc memory 0000000074f264cf
[415631.938189] alloc_disk_memory 16777216 address 00000000937e94d7
[415631.938189] alloc_disk_memory 16777216 address 00000000937e94d7
[415631.938189] xrd_alloc returns 00000000f525525b (00000000937e94d7)
[415631.938189] After xrd_alloc memory 00000000937e94d7
[415631.938189] adding_disk 0
[415632.474128] open disk
[415632.474132] revalidate
[415632.474199] copy_from_xrd retorna -19
...
```

Els missatges poden ser diferents, depenent del quadrimestre, ja que el codi del kernel canvia sovint.

Els valors negatius (errors) de lectura/escriptura són correctes, ja que això és el que tornen les rutines de lectura/escriptura que tenim a hores d'ara. I la sortida al `dmesg` pot continuar amb:

```
...
[415632.474772] close disk
[415632.477773] added_disk 0
[415632.477773] adding_disk 1
[415632.477773] open disk
[415632.477773] revalidate
[415632.477773] copy_from_xrd retorna -19
[415632.477773] copy_from_xrd retorna -19
[415632.477773] copy_from_xrd retorna -19
[415632.477773] close disk
[415632.477773] added_disk 1
[415632.477773] region range 8
[415632.477773] region added
[415632.477773] init returns 0
```

1 Si no existeix /dev/ram0, el podeu crear amb aquesta seqüència de comandes:

```
sudo modprobe rd       # rd és el nom del mòdul que implementa el RAMDisk oficial de Linux
sudo mknod -m 660 /dev/ram0  b 1 0
sudo chown root.disk /dev/ram0
sudo dd if=/dev/zero of=/dev/ram0 bs=1k count=4k
```

```
[415632.494449] open disk
[415632.494942] open disk
[415632.580092] close disk
[415632.589679] close disk
```

2) A banda de fer servir el dmesg, podeu fer, a un altre terminal:

```
$ tail -f /var/log/syslog
```

Llegiu del disc (usant `cat`, `dd` o `od`) i comproveu que la comanda us dóna l'error de lectura/escriptura que tornen actualment les operacions, per exemple:

```
$ cat /dev/xrd1
copy_to_xrd retorna -19
```

3) Escriviu al disc, i torneu a comprovar que la comanda que feu servir us dóna un error d'I/O:

```
$ dd of=/dev/xrd0 if=/dev/zero
copy_to_xrd retorna -19
1+0 records in
0+0 records out
...
```

4) També la comanda `fdisk` torna el mateix error:

```
$ fdisk /dev/xrd0
...
open disk
 xrd_ioctl cmd 21297 argd 0 (BLKFLSBUF 4705)
xrd_getgeo returns -5
copy_from_xrd retorna -19
copy_from_xrd retorna -19
xrd_ioctl cmd 21297 argd 0 (BLKFLSBUF 4705)
copy_from_xrd retorna -19
...
close disk
```

En aquest punt no podem fer més proves fins a tenir més operacions (lectura o escriptura) del disc ja programades.

Operació per retornar la geometria

Per retornar la geometria del disc a la vostra funció - aquesta funció sembla que actualment no es fa servir:

```
int xrd_getgeo(struct block_device * bdev, struct hd_geometry *geo);
```

cal accedir al vostre disc (bdev->bd_disk->private_data, de tipus xrd_struct) i agafar la mida (camp *size*). Llavors, cada camp de l'estructura hd_geometry (<linux/hdreg.h>):

```
  struct hd_geometry {
         unsigned char heads;
         unsigned char sectors;
         unsigned short cylinders;
         unsigned long start;
   };
```

s'omple amb:

heads= 32

sectors= 128

cylinders= <mida del disc> / geo->heads/geo->sectors/SECTOR_SIZE

start= 0

i no oblideu retornar el valor zero (0) enlloc de l'error actual, per indicar que tot ha anat bé. Aquesta funció la podrem provar més endavant.

Operació del lectura del disc

Les operacions de lectura (i escriptura) del disc arriben a les funcions `xrd_submit_bio` i `xrd_rw_page`, tal i com s'ha definit a la inicialitació del disc (podeu veure-ho al fitxer `interface.c`, a la taula d'operacions `xrd_fops`), amb els punters a les operacions:

```
.submit_bio = xrd_submit_bio,
.rw_page    = xrd_rw_page,
```

Aquestes funcions parteixen les peticions de lectura i escriptura en elements de com a màxim el tamany d'una pàgina de memòria virtual (4 KBytes). Veieu-ho en el bucle implementat així a `xrd_submit_bio`:

```
bio_for_each_segment(bvec, bio, iter) {
        unsigned int len = bvec.bv_len;
        err = xrd_do_bvec(xrd, bvec.bv_page, len, bvec.bv_offset, bio_op(bio), sector);
        if (err) goto io_error;
        sector += len >> SECTOR_SHIFT;
}
```

La funció `xrd_do_bvec` és la que classifica les operacions entre lectures i escriptures, prepara el mapeig de les dades de l'usuari a l'espai de sistema amb `kmap_atomic` i `flush_dcache_page`, i crida a `copy_from_xrd` per les lectures i `copy_to_xrd` per les escriptures.

El fet que `kmap_atomic` mapeji les dades de l'usuari a l'espai de sistema fa que no haguem d'usar les funcions que vèiem al dispositiu de caràcter `copy_to_user` i `copy_from_user`. Ara n'hi haurà prou amb usar la funció `memcpy`, que tenim també implementada a dins de Linux.

Per implementar la funció de lectura del disc a la pràctica, cal que implementeu la funció:

```
int copy_from_xrd(void *dst, struct xrd_device *xrd, sector_t sector, size_t n);
```

Aquesta funció rep:

dst: adreça en l'espai del kernel on escriure la informació llegida del disc, i que se li retornarà a l'usuari

xrd: punter a l'estructura del nostre disc

sector: sector del disc d'on s'ha de començar a llegir

n: numero de bytes que volem llegir

La funció ha de calcular la posició dins la taula de caràcters `xrd->disk_memory` on ha de començar la lectura. Fixeu-vos que la posició s'haurà de calcular a partir del `sector` rebut per paràmetre, multiplicat per la mida d'un sector (definit com a SECTOR_SIZE, a `interface.h`) i sumar aquest valor a `xrd->disk_memory`.

Un cop calculada aquesta posició dins les dades del disc, podeu fer el `memcpy` per copiar `n` bytes d'informació llegits a partir d'aquesta posició, a l'adreça `dst`.

Finalment, `copy_from_xrd` retornarà zero (0) en cas que tot hagi anat bé. Altrament retornarà el codi negatiu de l'error.

Proves de lectura del disc

Un cop implementada la funció de lectura del disc, inserteu el mòdul i feu les següents proves:

5) Llegir del disc, com habitualment, amb `dd`, `cat` o `od`, per veure el contingut. Observeu que aquest ha de coincidir amb el patró de dades que heu fet servir per inicialitzar-lo. Comenteu la sortida que obteniu a la taula-resum que us demana la pràctica (veieu secció d'Entrega, al final del document):

```
$ od -x /dev/xrd1
0000000 0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
0000020 1110 1312 1514 1716 1918 1b1a 1d1c 1f1e
...
```

Aquesta informació s'obté havent inicialitzat els caràcters de l'àrea `disk_memory` amb el patró 0, 1, 2... 254, 255, 0, 1, 2... i així repetit fins al final.

6) Ara la comanda *fdisk* ha de funcionar, llegint del disc correctament, però donant un error a l'escriure la taula de particions, feu:

```
$ fdisk /dev/xrd1
...
Command (m for help): n                 # creeu una partició
...
Command (m for help): w                 # sortir i escriure la taula de particions
The partition table has been altered!
Calling ioctl() to re-read partition table
Error closing file                      # aquest deu ser l'error en l'escriptura
```

Operació d'escriptura del disc

Per implementar la funció d'escriptura al disc, cal que implementeu la funció:

```
int copy_to_xrd(struct xrd_struct *xrd, void * src, sector_t sector, size_t n);
```

Aquesta funció rep:

xrd: punter a l'estructura del nostre disc

src: adreça en l'espai del kernel d'on llegirà la informació de l'usuari per escrire-la al disc

sector: sector del disc d'on s'ha de començar a escriure

n: numero de bytes que volem escriure

La funció ha de calcular la posició dins la taula de caràcters `xrd->disk_memory` on ha de començar l'escriptura. Fixeu-vos que la posició s'haurà de calcular a partir del `sector` rebut per paràmetre, i multiplicat per la mida d'un sector (definit com a SECTOR_SIZE, a `interface.h`).

Un cop calculada aquesta posició dins les dades del disc, podeu fer el `memcpy` per copiar `n` bytes d'informació llegits a partir de l'adreça `src`, a la posició calculada del disc.

Finalment, `copy_to_xrd` retornarà zero (0) en cas que tot hagi anat bé. Altrament retornarà el codi negatiu de l'error.

Proves completes de funcionament del disc

Inserteu el mòdul amb el vostre disc. Les proves següents intenten anar de poca a major complexitat, per veure si tota la funcionalitat del vostre disc és correcte. **Recordeu que si alguna de les proves falla haureu d'analitzar quin és el problema i que per a fer-ho us anirà bé**, dins les possibilitats que dóna, **veure el contingut del `dmesg`, amb els missatges del kernel.**

7) Escriptura a disc. Feu:

```
$ cp interface.c /dev/xrd0    #copiem el fitxer interface.c a l'inici del disc
$ cat /dev/xrd0 | more
                # heu de veure les mateixes dades que té el fitxer interface.c
#include <linux/module.h>
...
$ cmp /dev/xrd0 interface.c
cmp: EOF on interface.c                # veure l'explicació a la pàg. següent
```

La darrera comanda (`cmp`) compara el contingut del disc, començant per la posició zero (0), amb el contingut del fitxer. Quan diu que s'ha trobat un End-Of-File (EOF) al fitxer, vol dir que la comparació ha estat correcta fins que s'ha acabat el fitxer. Observeu que el disc no té una marca de final de fitxer en aquest punt. Això vol dir que el disc continuaria donant-nos informació (que hauria de ser la continuació de la seqüencia 0, 1, 2... que hi havia d'origen, per allà on anés més enllà de la informació copiada d'interface.c.

8) Useu l'`fdisk` i creeu una partició que ocupi tot el disc:

```
$ fdisk /dev/xrd0

...

Command (m for help): n           # i creeu la partició

...

Command (m for help): w

...

Syncing disks.
```

Ara l'escriptura de la taula de particions hauria d'anar bé i no donar l'error d'abans (Error closing file). En el seu lloc dóna el "Syncing disks." Després de modificar la taula de particions, el sistema actualitzarà el `/dev`, incloent-hi l'entrada corresponent a la nova partició, `/dev/xrd0p1`.

9) Ara l'`fdisk` ha de poder llistar la partició creada:

```
$ fdisk -l /dev/xrd0
Disk /dev/xrd0: 16 MB, 16777216 bytes
32 heads, 128 sectors/track, 8 cylinders, total 32768 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x26c6289f

     Device Boot      Start         End      Blocks   Id  System
/dev/xrd0p1            2048       32767       15360   83  Linux
```

Comproveu també que s'ha creat el fitxer de dispositiu `/dev/xrd0p1`:

```
$ ls -ltr /dev/xrd*
brw-rw---- 1 root disk 240, 4 Nov  4 15:41 /dev/xrd1
brw-rw---- 1 root disk 240, 1 Nov  4 15:52 /dev/xrd0p1
brw-rw---- 1 root disk 240, 0 Nov  4 15:52 /dev/xrd0
```

10) Useu `badblocks` per comprovar la partició dins el vostre disc:

```
$ badblocks /dev/xrd0p1                          # no hauria de donar cap sortida
$ badblocks -w -v /dev/xrd0p1                    # verbose i esborrant el contingut
Checking for bad blocks in read-write mode
From block 0 to 15359
Testing with pattern 0xaa: done
Reading and comparing: done
Testing with pattern 0x55: done
Reading and comparing: done
Testing with pattern 0xff: done
Reading and comparing: done
Testing with pattern 0x00: done
Reading and comparing: done
Pass completed, 0 bad blocks found. (0/0/0 errors)
```

11) Construiu un sistema de fitxers vfat a /dev/xrd0p1

```
$ mkfs -t vfat /dev/xrd1                         # No hauria de donar errors
mkfs.vfat 3.0.11 (24 Dec 2010)
```

En particular no hauria de donar l'error "**unable to get drive geometry, using default 255/63**", si heu implementat correctament la funció `xrd_getgeo`.

12a) Construiu un sistema de fitxers ext2 a /dev/xrd1

```
$ mkfs -t ext2 /dev/xrd1                         # No hauria de donar errors
mke2fs 1.42.6 (21-Sep-2012)
Discarding device blocks: done
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
4096 inodes, 16384 blocks
819 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=16777216
2 block groups
8192 blocks per group, 8192 fragments per group
2048 inodes per group
Superblock backups stored on blocks:
        8193

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

No hauria de donar cap error.

12b) Verifiqueu el sistema ext2 que heu creat a 12a)

```
$ fsck /dev/xrd1
fsck from util-linux 2.20.1
e2fsck 1.42.5 (29-Jul-2012)
/dev/xrd1: clean, 11/4096 files, 661/16384 blocks
```

No hauria de donar cap error.

13) Munteu la partició acabada de fer:

```
$ mkdir /mnt/point                              # creem el punt de muntatge
$ mount /dev/xrd1 /mnt/point                    # muntem la partició
$ ls -l /mnt/point                              # comprovem que hi ha un ext2
total 12
drwx------ 2 root root 12288 Nov  4 16:07 lost+found/
```

La sortida del dmesg hauria de ser semblant a aquesta (si teniu un printk de les adreces on feu les còpies amb el memcpy):

```
[ 3218.795924] open disk
[ 3218.805415] read_from_disk ffffc90005477400 -> ffff880016745400 (1024)
[ 3218.810918] read_from_disk ffffc90005477800 -> ffff880016745800 (1024)
[ 3218.818509] read_from_disk ffffc90005488000 -> ffff880017913000 (1024)
[ 3218.824500] write_to_disk  ffff880016745400 -> ffffc90005477400 (1024)
```

14) Traieu el mòdul amb `rmmod`. Per què no el podeu treure, mentre el dispositiu està muntat? Pista, veure la sortida del `lsmod`.

15) Finalment, mireu l'script **check-disk.sh** per veure què fa i executeu-lo de les següents maneres:

```
$ ./check-disk.sh ext2
$ ./check-disk.sh vfat
$ ./check-disk.sh ext3
$ ./check-disk.sh ext4
$ ./check-disk.sh reiserfs       # aquesta segurament fallarà
$ ./check-disk.sh reiser4        # aquesta també fallarà, per què?
```

Segurament no totes sis acabaran bé amb el missatge: "Test SUCCESSFUL". Mireu quina pot ser la causa de l'error i si el podeu corregir.

<opcional> 16) L'script **check-partition.sh** hauria de funcionar de la mateixa manera, però creant una partició amb fdisk a dins del disc primer. Comproveu-ho.

Entrega, a través del Racó:

- un resum amb els resultats de les proves que heu fet amb el *ramdisk*. Podeu fer-lo en forma de taula:

Prova	**Comentari (ok, o el motiu pel qual no dóna el resultat esperat)**
1	
2	
3	
...	
...	

- la sortida de ./check-disk.sh amb els 6 tipus de sistemes de fitxers ext2, vfat, ext3, ext4, reiserfs i reiser4 (no tots han de funcionar correctament, consulteu amb els professors els possibles motius).

- el codi amb els fitxers que implementen el ramdisk, el `Makefile` i el shell script `check-disk.sh` per comprovar que funciona.