Diseño de Sistemas Operativos

RAMDISK

Anastasia Ramos Mayor
Ellishia Owens Cruz
Facultad de Informática de la ULPGC

Definición

- Zona de memoria RAM configurada para simular disco duro. Consiste, en otras palabras, en reservar una cierta cantidad de memoria y acceder a ella como si fuese una unidad de disco.
  - Pierden su contenido al apagar el sistema. Es necesario cargar los datos al iniciar la sesión.
  - Se accede como un disco duro normal, pero con la velocidad de una memoria RAM.
  - Se configura como un dispositivo en modo bloque.

Definición: uso

- Cuando un grupo de archivos va a ser accedido de forma intensiva, guardar esos archivos en memoria incrementará la velocidad de acceso debido a la gran diferencia de velocidad que existe entre la memoria y un disco duro.
- Lo usa initrd para el arranque del Sistema cargando en el dispositivo los módulos principales.

Propiedades

- RAM disk crece dinámicamente según se va necesitando espacio. Esto es posible gracias a la estructura de árbol de páginas de la que hace uso, que a su vez emplea la estructura buffer cache.
- Los bufferes se marcan como dirty para que el sistema de memoria virtual no intente acceder a ellos.
- RAM disk soporta hasta 16 dispositivos RAM por defecto pudiendo configurarse para un número ilimitado mediante el símbolo de configuración BLK_DEV_RAM_COUNT. (Si se modifica este parámetro hay que recompilar el kernel)
- Es capaz de cargar imágenes comprimidas de disco RAM permitiendo contener más programas en una instalación promedia o en un disco de arranque.

Parámetros

- ramdisk_size=N

Especifica el tamaño en bytes de los discos RAM. Por defecto este valor es 4096 (4MB).

- ramdisk_blocksize= N

Especifica el tamaño en bytes de los bloques, es decir, el tamaño de bloques del dispositivo RAM disk. Por defecto este parámetro tiene valor 1024 (1MB).

Implementación

- La implementación del manejador de este tipo de dispositivos se localiza en el fichero **/drivers/block/brd.c**
- Además de encontrarse la especificación para esta clase de discos, se ha incluido la implementación de manejador de dispositivos **loop** debido a las similitudes entre ambos.
- El loop es un controlador de dispositivos que permite que un archivo imagen pueda ser montado como un dispositivo de bloque normal. Esto quiere decir, que podemos utilizar archivos imágenes como ”discos duros” virtuales 100% funcionales, pudiendo leer, escribir, sobreescribir y borrar de la misma manera.

Estructuras (BRD_DEVICE)

```
35   struct brd_device {
36        int             brd_number;
37        int             brd_refcnt;
38        loff_t           brd_offset;
39        loff_t           brd_sizelimit;
40        unsigned        brd_blocksize;

42        struct request_queue    *brd_queue;
43        struct gendisk          *brd_disk;
44        struct list_head        brd_list;
45

50        spinlock_t              brd_lock;
51        struct radix_tree_root  brd_pages;
52};
```

Estructuras

Esta estructura indica al sistema del núcleo encargado de manejar los bloques de memoria las operaciones que debe realizar en cada caso.

```
376 static struct block_device_operations brd_fops = {
377         .owner =                THIS_MODULE,
378         .ioctl =                brd_ioctl,
379#ifdef CONFIG_BLK_DEV_XIP
380         .direct_access =        brd_direct_access,
381#endif
382};
```

Estructuras(Radix_tree)

- Estructura utilizada para almacenar la información contenida en el dispositivo virtual.
- Implementa un árbol de ”páginas”
- Las funciones importantes son las siguientes:
- Reserva de espacio para nuevas inserciones deshabilitando el preempt:

```
int radix_tree_preload(gfp_t gfp_mask)
```

- Insercción de un elemento en el árbol apuntado por root en la posición index

```
int radix_tree_insert(struct radix_tree_root *root,  unsigned long index, void *item)
```

- Búsqueda en el radix tree

```
void *radix_tree_lookup(struct radix_tree_root *root, unsigned long index)
```

Estructuras (Radix_tree II)

- Búsqueda múltiple en el radix tree

```
unsigned int radix_tree_gang_lookup(struct radix_tree_root *root, void
    **results,  unsigned long first_index, unsigned int max_items)
```

- Terminación de la reserva de espacio habilitando el preempt

```
static inline void radix_tree_preload_end(void)
```

- Eliminar un elemento del árbol apuntado por la raíz *root

```
void *radix_tree_delete(struct radix_tree_root *root, unsigned long index)
```

Estructuras (BIO, BIO_VEC)

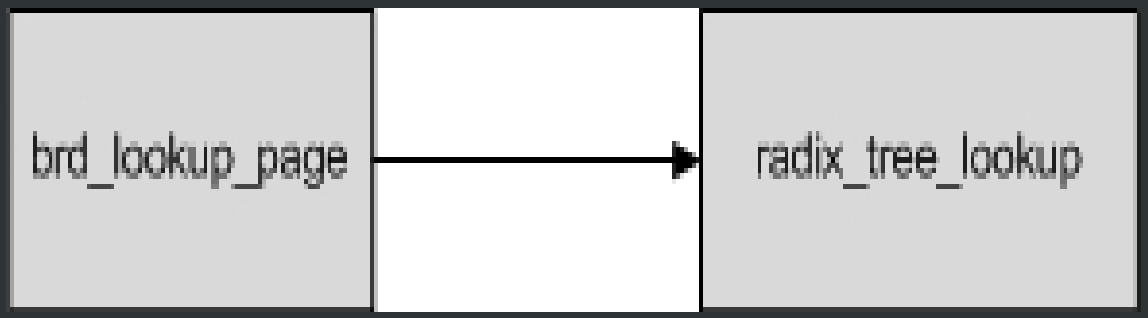
- La bio es una estructura que presenta la operación como lista de segmentos
- Los campos relevantes de la bio:
- bi_dev # dispositivo de bloques asociado a la petición
- bi_rw #tipo de petición
- bi_io_vec #vector de estructuras bio_vec

- Cada segmento de la estructura bio es una estructura bio_vec agrupados en una lista bi_io_vec

Funciones

Esta función se encarga de buscar una página determinada por el parámetro sector, en el dispositivo apuntado por *brd.

static struct page *brd_lookup_page(struct brd_device *brd, sector_t sector)

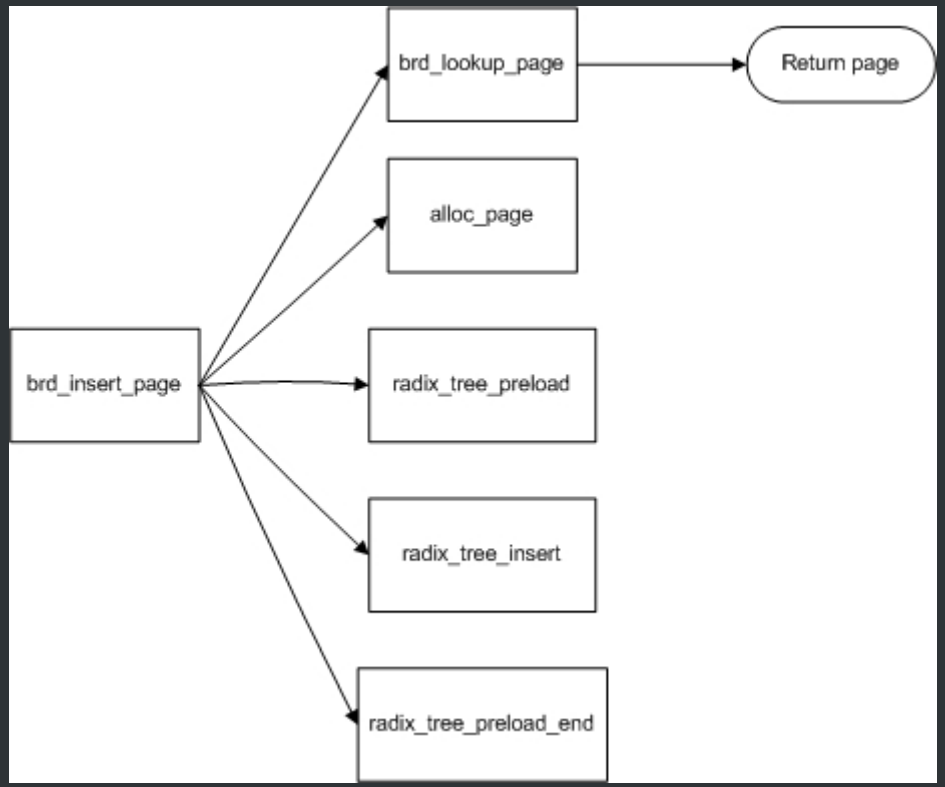


Codigo

Funciones

Busca una página. Si no existiera la crea vacia y la inserta en el árbol de páginas.

static struct page *brd_insert_page(struct brd_device *brd, sector_t sector)

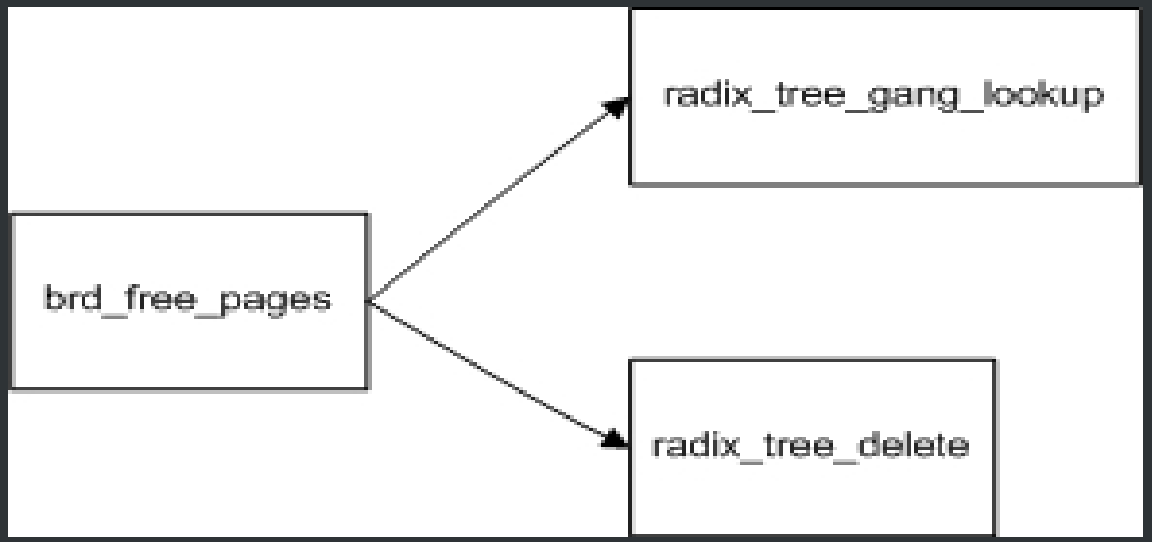


Codigo

Funciones

Liberar todas las páginas almacenadas y el árbol (radix tree). Esta función sólo debe usarse si no hay más usuarios del dispositivo

static void brd_free_pages(struct brd_device *brd)

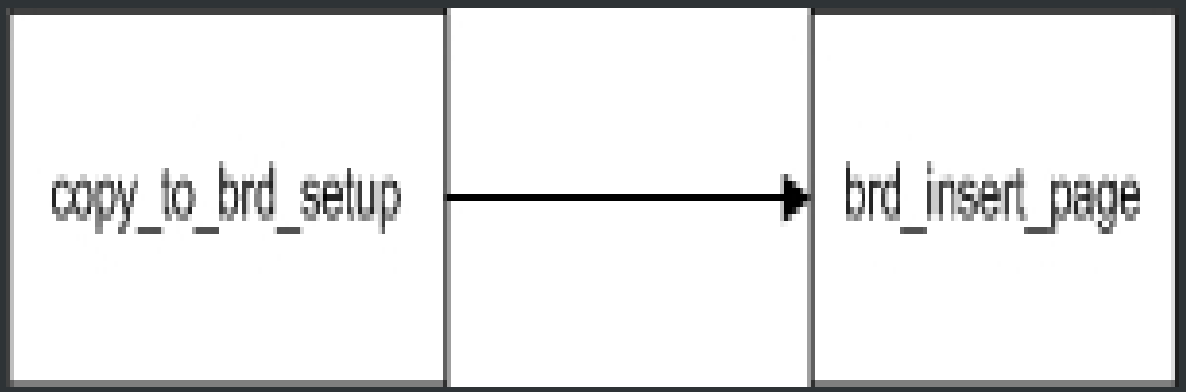


Codigo

Funciones

Preparar el dispositivo para realizar una copia.Introduce tantas páginas nuevas en el árbol como sea necesario para llegar al tamaño indicado (n)

static int copy_to_brd_setup(struct brd_device *brd, sector_t sector, size_t n)



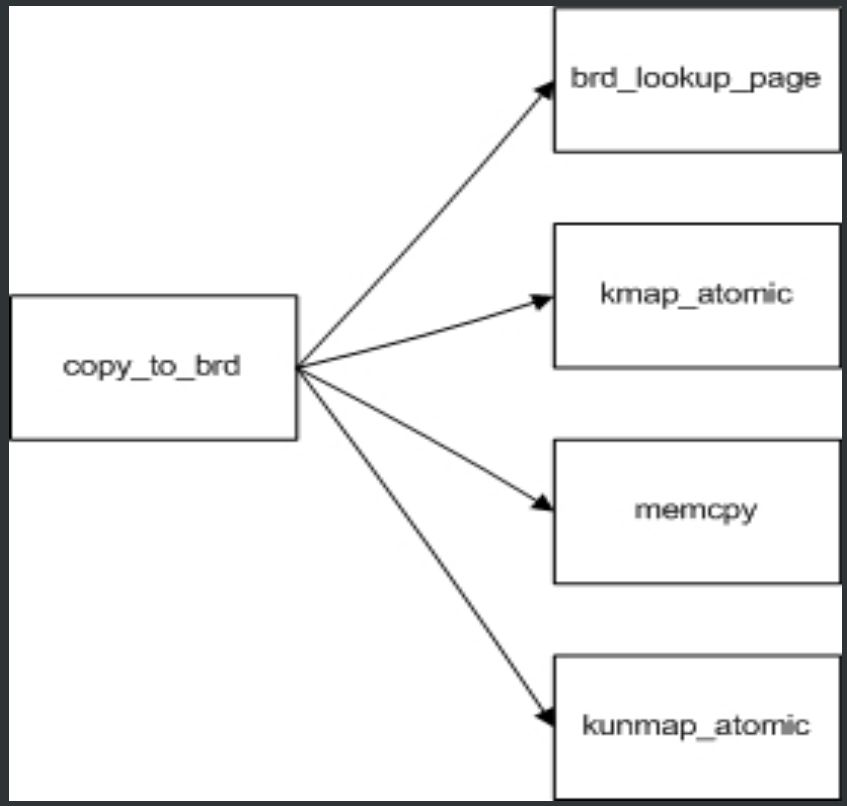
Debe llamarse antes de llamar a copy_to_brd.

Codigo

Funciones

Copia n bytes de scr en el brd empezando en sector.

static void copy_to_brd(struct brd_device *brd, const void *src, sector_t sector, size_t n)

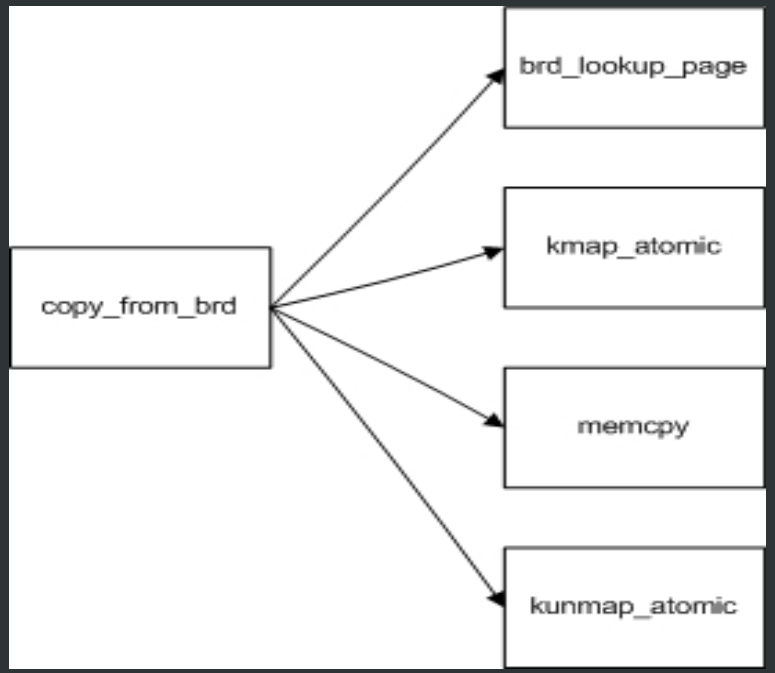


Codigo

Funciones

Copia n bytes a dst desde el brd empezando en el indice indicado por sector

static void copy_from_brd(void *dst, struct brd_device *brd, sector_t sector, size_t n)

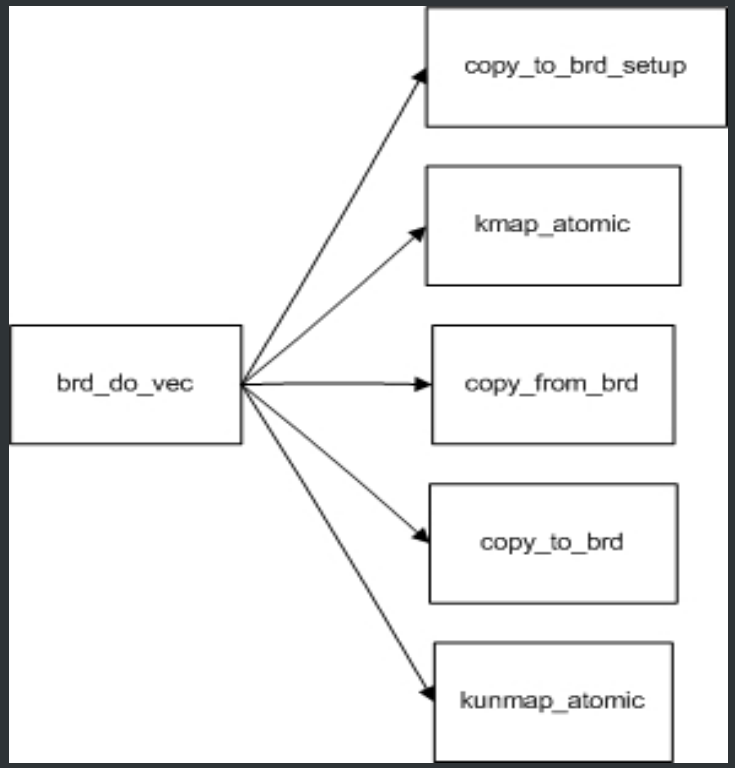


Codigo

Funciones

Ejecutar una operación de entrada/salida en función de rw, sobre el dispositivo brd, empezando en sector

static int brd_do_bvec(struct brd_device *brd, struct page *page
unsigned int len, unsigned int off, int rw, sector_t sector)

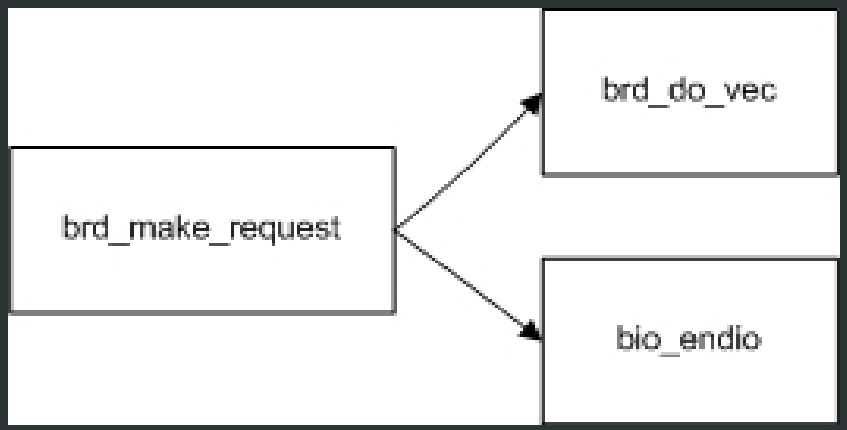


Codigo

Funciones

Una operación de entrada/salida se solicita a través de esta función. Es una especie de interfaz para la llamada al brd_do_bvec.

static int brd_make_request(struct request_queue *q,
struct bio *bio)

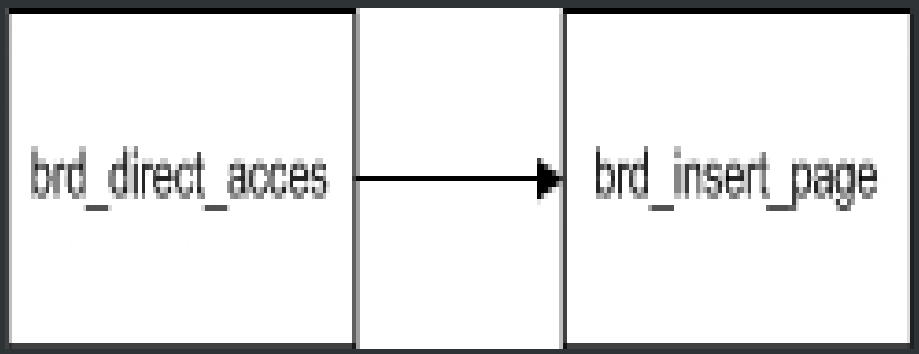


Codigo

Funciones

Esta función define el acceso directo. La opción de configuración CONFIG_BLK_DEV_XIP activada permite XIP filesystems en un dispositivo RAM, es decir, no es necesario el mapeo para las operaciones tipo entrada/salida ni utilizar funciones de copia.

```
#ifdef CONFIG_BLK_DEV_XIP
static int brd_direct_access (struct block_device *bdev, sector_t
sector,  unsigned long *data)
```



Codigo

Funciones

Implementa la llamada al sistema ioctl

static int brd_ioctl(struct inode *inode, struct file *file,
unsigned int cmd, unsigned long arg)

Si el comando cmd especificado es diferente de BLKFLSBUF devuelve un error indicando comando no válido. En otro caso libera la memoria asociada al inode pasado y elimina el dispositivo RAM para lo que libera su árbol de páginas de memoria.

Configuración de parámetros

Se establece la configuración de la RAM disk que se pretende crear.

```
387static int rd_nr;
388int rd_size = CONFIG_BLK_DEV_RAM_SIZE;
389module_param(rd_nr, int, 0);
390MODULE_PARM_DESC(rd_nr, "Maximum number of brd devices");
391module_param(rd_size, int, 0);
392MODULE_PARM_DESC(rd_size, "Size of each RAM disk in kbytes.");
393MODULE_LICENSE("GPL");
394MODULE_ALIAS_BLOCKDEV_MAJOR(RAMDISK_MAJOR);
```

Inicialización

Si no está definida la macro que habilita la carga modular del sistema se hace uso de las siguientes funciones de inicialización

```
396#ifndef MODULE
398static int __init ramdisk_size(char *str)
399{
400         rd_size = simple_strtol(str, NULL, 0);
401         return 1;
402}
403static int __init ramdisk_size2(char *str)
404{
405         return ramdisk_size(str);
406}
407__setup("ramdisk=", ramdisk_size);
408__setup("ramdisk_size=", ramdisk_size2);
409#endif
```

Ejemplo de uso

- Decidir el tamaño de RAM disk que se desea, por ejemplo 2MB

```
sudo dd if=/dev/zero of=/dev/ram0 bs=1k count=2048
```

- Crear el sistema de ficheros, en este caso ext2fs

```
sudo mke2fs -vm0 /dev/ram0 2048
```

- Montar la partición del sistema de fichero

```
sudo mount /dev/ram0 /home/aski/Desktop
```

- Copiar los datos que se quieran cargar

```
sudo cp /home/aski/ramdisk.pdf /home/aski/Desktop/ramdisk.pdf
```

- Desmontar la partición
- YA ESTÁN LOS FICHEROS EN MEMORIA

RAM DISK

FIN