Taller de Sistemas Operativos

Dispositivos de E/S orientados a caracteres (char devices)

Char Devices

- Los char devices se caracterizan por el acceso secuencial de trozos de información de tamaño variable
- El más común de los char devices son las terminales
- Otros dispositivos son impresoras, mouse, puertos seriales
- En estos los datos se acceden como un stream de datos secuenciales; de a un byte por vez
- El teclado por ejemplo es un character device; El device provee un stream de caracteres que el usuario tipea.

Char Devices

- El manejo de los char devices es más sencillo que de los block devices
- Hay char devices con un protocolo complejo y otro que apenas lee de un par de IO/ports.
- El driver de un character device se representa con una estructura *cdev*

Type	Field	Description
struct kobject	kobj	Embedded kobject
struct module *	owner	Pointer to the module implementing the driver, if any
struct file_operations *	ops	Pointer to the file operations table of the device driver
struct list_head	list	Head of the list of inodes relative to device files for this character device
dev_t	dev	Initial major and minor numbers assigned to the device driver
unsigned int	count	Size of the range of device numbers assigned to the device driver

Char Devices

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned
    int (*check_flags)(int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsig
    ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigr
    int (*setlease)(struct file *, long, struct file_lock **);
    long (*fallocate)(struct file *file, int mode, loff_t offset,
              loff_t len);
};
```

Char devices

- `cdev_alloc ()`
  - Crea una estructura *cdev* e inicializa el k_object embebido.
- `cdev_add ()`
  - Inicializa los campos *dev* y *count* y luego invoca *k_object_map()*
- El DDM define un dominio de mapeo de k_objects para los c devices representado por el registro *k_obj_map* y referenciado por la variable global *cdev_map.*
- Incluye una hash table de 255 entradas indexada por el major number correspondiente a los intervalos. Guarda objetos de tipo *probe.*

Type	Field	Description
`struct probe *`	`next`	Next element in hash collision list
`dev_t`	`dev`	Initial device number (major and minor) of the interval
`unsigned long`	`range`	Size of the interval
`struct module *`	`owner`	Pointer to the module that implements the device driver, if any
`struct kobject *(*) (dev_t, int *, void *)`	`get`	Method for probing the owner of the interval
`int (*)(dev_t, void *)`	`lock`	Method for increasing the reference counter of the owner of the interval
`void *`	`data`	Private data for the owner of the interval

Asignación de Device Numbers

- Para administrar cuales character devices están en uso, el kernel usa un hash *chrdevs*
- Dos intervalos pueden tener el mismo major pero no pueden superponerse
- Cada elemento del hash es de tipo *char_device_struct*

Type	Field	Description
unsigned char_device_struct *	next	The pointer to next element in hash collision list
unsigned int	major	The major number of the interval
unsigned int	baseminor	The initial minor number of the interval
int	minorct	The interval size
const char *	name	The name of the device driver that handles the interval
struct file_operations *	fops	Not used
struct cdev *	cdev	Pointer to the character device driver descriptor

Asignación de Device Numbers

- Tres maneras
  - *register_chrdev_region( )*
    - Device number inicial (major,minor)
    - Rango de device number
    - Nombre del device driver
    - No llama a *cdev_add ()*
  - *alloc_chrdev_region( )*
    - Device number inicial (minor)
    - Rango de device number
    - Nombre del device driver
    - No llama a *cdev_add ()*
  - *register_chrdev( )*
    - Major number
    - Nombre del device driver
    - Puntero a *fops*
    - Llama a *cdev_add ()*

Asignación de Device Numbers

- *register_chrdev_region( )*
  - Verifica si el pedido cubre varios “major numbers”
  - En caso afirmativo, arma cada rango y termina llamando a *__register_chrdev_region( )* para cada un de ellos

- *alloc_chrdev_region( )*
  - Similar a la anterior pero es usada para alocar un major number dinámicamente.
  - Termina invocando a *__register_chrdev_region( )*

Asignación de Device Numbers

- *__register_chrdev_region()*
  - Aloca una estructura *char_device_struct* y la rellena con ceros
  - Si el major number pasado como parámetro es cero, se pide asignación dinámica. Se busca una entrada vacía en la tabla de hash.
  - Inicializa los campos de la estructura *char_device_struct* con el nro inicial del intervalo, el tamaño y el nombre
  - Ejecuta la función de hash para saber según el major number, el índice
  - Recorre el bucket del hash para ubicar la estructura *char_device_struct* . Si se superpone con una existente, retorna error.
  - Inserta la estructura *char_device_struct* en el lugar econtrado
  - Retorna un puntero a la estructura *char_device_struct*

Asignación de Device Numbers

- *register_chrdev( )* es utilizada cuando se requiere un único major y los 255 minor.
- *register_chrdev( )*
  - *Invoca a __register_chrdev_region( ) con el intervalo requerido. Si retorna error se aborta*
  - Aloca una estructura *cdev*
  - Asigna el estado del k_object a *ktype_cdev_dynamic*
  - Asigna el campo *owner* *con* *fops->owner*
  - Asigna el campo *ops* con el contenido de *fops*
  - Copia el nombre del dispositivo dentro del campo *name* del k_object embebido
  - Asigna el campo *cdev* del descriptor *char_device_struct* (retornado en el 1er paso) con la descripción del descriptor *cdev* del dev driver
  - Retorna el major number del intervalo asignado

Accediendo un Char Device

- Open sobre un char device file
- Se llama a *dentry_open()* que llama a *chrdev_open()*
- *chrdev_open()* recibe como parámetro el *inode* y el *filp* (file descriptor) del archivo accedido
  1. Verifica el puntero *inode->i_cdev*. Si es *<> NULL* el inodo ya ha sido accedido. Incrementa la ref count del descriptor *cdev* y va al paso 6
  2. Invoca a *kobj_lookup( )* para buscar el intervalo. Si no existe retorna error; sino, calcula la dirección del desc cdev
  3. Asigna al campo *inode->i_cdev* del inodo la dirección de *cdev*
  4. Traduce el *inode->i_cindex* al índice relativo a dentro del intervalo del device driver (0 para el primer minor ……)
  5. Agrega el objeto del inodo en la lista apuntada por el campo *list* del descriptor *cdev*
  6. Inicializa el puntero *filp->f_ops* con el contenido del campo *ops* del descriptor *cdev*
  7. Si el método *filp->f_ops->open* se encuentra definido lo ejecuta
  8. Termina la ejecución retornando 0 (éxito)

Buffering en Char Devices

- Uso de DMA para tranferir los datos
- Uso de un buffer circular de dos o más elementos. Cuando ocurre una interrupción indicando que hay datos disponibles el handler de la interrupción avanza el puntero. Cuando el driver copia el elemnto en user space, se libera el elemento del buffer circular.