Utilización De Una Función Del Sistema Operativo Por Un Usuario

s. candela

Llamada al sistema: open

Los usuarios utilizan las funciones del sistema operativo mediante las llamadas al sistema (~200 en versión 2 de Linux), definidas en el sistema por un nombre y un número, (open, 5).

- El nombre de la llamada se encuentra definido en el archivo ensamblador arch/i386/kernel/entry.s

  .long SYMBOL_NAME (sys_open) /* 5 */
- El número en include/asm/unistd.h
- 1. Un usuario en su programa en C invoca una función de la biblioteca del sistema

  fd = open ("/dev/hda", O_RDONLY);
- 2. La biblioteca en C, (libc, libm) proporciona funciones que invocan a las llamadas al sistema open -> syscall -> INT 0x80 (parámetros de la llamada en los registros del procesador)

  INTERRUPCIÓN SOFTWARE paso de modo usario a modo NÚCLEO
- 3. mediante las tablas IDT + GDT se ejecuta la función sys_call definida en arch/i386/kernel/entry.s

  sys_call utiliza el número de llamada al sistema (open, 5) transmitido en el registro eax para buscar en la tabla sys_call_table la dirección de la llamada al sistema
- 4. las llamadas tienen la sintaxis: sys_nombre (argumentos)

  int sys_open (const char *filename, int flags, int mode)
- 5. cuando la llamada termina vuelve el control a sys_call y este devuelve el control al usuario y se retorna a modo USUARIO.

Programa para leer el primer sector del disco duro. (según la idea de Andries Brouwer, aeb@cwi.nl)

- Abre el fichero especial “/dev/hda”, asociado al primer disco duro IDE.
- Lee en la variable “buf” el primer sector del disco

```
#include <unistd.h> /* números de las llamadas al
sistema */
#include <fcntl.h>
int main(){
        int fd;
        char buf[512];
        fd = open("/dev/hda", O_RDONLY);
        if (fd >= 0)
                read(fd, buf, sizeof(buf));
        return 0;
}
```

open

- El fuente de la llamada al sistema sys_open se encuentra en fs/open.c, las líneas maestras de la llamada son:
- (1) Obtiene memoria en el núcleo y pasa el nombre del fichero al núcleo.
- (2) Obtiene un descriptor de fichero no utilizado fd.
- (3) Realiza el open, busca el inode asociado al fichero.
- (4) Almacena información del paso (3) en la ranura del programa que llama.
- (5) Libera el espacio de memoria en el núcleo reservada en (1) devuelve el descriptor de fichero fd

sys_open

- Veamos los fuentes, quitando las líneas no esenciales, el manejo de errores y de bloqueo.

```
int sys_open(const char *filename, int flags, int mode) {
        char *tmp = getname(filename);                  /*(1)*/
        int fd = get_unused_fd();                       /*(2)*/
        struct file *f = filp_open(tmp, flags, mode);   /*(3)*/
        fd_install(fd, f);                              /*(4)*/
        putname(tmp);                                   /*(5)*/
        return fd;
}
```

(1) getname()

- La función getname() se encuentra en fs/namei.c. Pasa parámetros al núcleo.
- asigna memoria en el núcleo para el nombre del fichero “/dev/hda” .
- copia el nombre del fichero desde el espacio del usuario al espacio del núcleo.

```
#define __getname()     kmem_cache_alloc(names_cachep, SLAB_KERNEL)
#define putname(name)   kmem_cache_free(names_cachep, (void *)(name))

char *getname(const char *filename) {
        char *tmp = __getname();        /* asigna memoria */

        strncpy_from_user(tmp, filename, PATH_MAX + 1);

        return tmp;}
```

(2) get_unused_fd()

- El procedimiento get_unused_fd() se encuentra en fs/open.c.
- current es un puntero que señala en la tabla de procesos (task struct) la ranura del proceso que hizo el open.
- la estructura files almacena la información de ficheros del proceso
  busca el primer descriptor de fichero libre
- lo marca como usado en la estructura files
- devuelve el descriptor de fichero encontrado

```
int get_unused_fd(void) {
        struct files_struct *files = current->files;
        int fd = find_next_zero_bit(files->open_fds,
                                    files->max_fdset, files->next_fd);
        FD_SET(fd, files->open_fds);    /* en uso ahora */
       files->next_fd = fd + 1;
        return fd;
}
```

(4) fd_install()

- El procedimiento fd_install() se encuentra en include/linux/file.h.
- Almacena información devuelta por filp_open() en la estructura files.

```
void fd_install(unsigned int fd, struct file *file) {

        struct files_struct *files = current->files;

         files->fd[fd] = file;
}
```

(3) filp_open()

- La function filp_open() se encuentra en fs/open.c.
- Realiza el trabajo principal de sys_open().
- El inode contiene la información que precisa el s.o. sobre un fichero.
- dentry, entrada a un directorio describe el nombre de un fichero, mediante el número del inode mas la ruta para encontrarlo.
- Utiliza la estructura nameidata que se encuentra definida en include/linux/fs.h. en la búsqueda del inode, almacena la entrada al directorio dentry y el sistema de fichero utilizado mnt que nos lleva a los procedimientos que manejan las estructuras, superbloque, inode, … de ese FS.

```
struct nameidata {
        struct dentry *dentry; /*entrada en el directorio*/
        struct vfsmount *mnt; /*tipo de sistema de ficheros utilizado*/
        struct qstr last; /* contiene el path */
};
```

filp_open()

- busca el inode asociado con el fichero mediante open_namei
- busca una entrada en la tabla filp, con dentry_open

```
struct file *filp_open(const char *filename, int flags, int mode) {
        struct nameidata nd;
        open_namei(filename, flags, mode, &nd);
        return dentry_open(nd.dentry, nd.mnt, flags);
}
```

open_namei()

- El procedimiento open_namei() se encuentra en fs/namei.c:
- llena campos de la estructura nameidata (nd->mnt y nd->dentry).
- realiza la búsqueda de la entrada de directorio mediante path_walk() .

```
open_namei(const char *pathname, int flag, int mode, struct nameidata *nd) {
        if (!(flag & O_CREAT)) {
                /* El caso mas simple – realizar una búsqueda sencilla. */
                if (*pathname == '/') {          /*directorio root*/
                        nd->mnt = mntget(current->fs->rootmnt);
                        nd->dentry = dget(current->fs->root);
                } else {                         /*directorio actual*/
                        nd->mnt = mntget(current->fs->pwdmnt);
                        nd->dentry = dget(current->fs->pwd);
                }
                path_walk(pathname, nd);
                /* Check permissions etc. */
                ...
                return 0;
        }
        ...
}
```

path_walk

- Busca en la cache de directorios recientemente usados la entrada de directorio mediante cached_lookup(). Si no encuentra la entrada.
- Busca la entrada de directorio en el sistema de ficheros mediante real_lookup(), que irá a disco.
- Al finalizar nd contiene la entrada al directorio, o lo que es lo mismo el inode con la información del fichero

```
path_walk (const char *name, struct nameidata *nd) {
        struct dentry *dentry;
        for(;;) {
                struct qstr this;
                this.name = next_part_of(name);
                this.len = length_of(this.name);
                this.hash = hash_fn(this.name);
                /* if . or .. then special, otherwise: */
                dentry = cached_lookup(nd->dentry, &this);
                if (!dentry)
                        dentry = real_lookup(nd->dentry, &this);
                nd->dentry = dentry;
                if (this_was_the_final_part)
                        return;
        }
}
```

dentry_open()

- encuentra una entrada vacía en la tabla “filp”.
- initializa la estructura “file”.

```
struct file *

dentry_open(struct dentry *dentry, struct vfsmount *mnt, int flags) {
        struct file *f = get_empty_filp();
        f->f_dentry = dentry;
        f->f_vfsmnt = mnt;
        f->f_pos = 0;
        f->f_op = dentry->d_inode->i_fop;
        ...
        return f;
}
```

Procedimientos para trabajar con el **FS**

- Cada sistema de fichero proporciona unas estructuras que contienen los procedimientos para trabajar con el:
- super_operations, procedimientos para trabajar con la estructura superbloque.
- file_operations, procedimientos para trabajar con la estructura file. inode_operations, procedimientos para trabajar con la estructura inode.
- address_space_operations direcciones de procedimientos que hacen otras cosas y a sí mas.
- Para un sistema de ficheros especifico, la dirección de la función de búsqueda real_lookup, se encuentra en la estructura inode_operations en el inode contenido en la entrada "dentry" del directorio para el que estamos realizando la búsqueda.
- Esta función especifica para un determinado sistema de ficheros, debe leer en el disco y buscar en el directorio para este fichero que estamos buscando.

```
struct dentry *
real_lookup(struct dentry *parent, struct qstr *name, int flags) {
        struct dentry *dentry = d_alloc(parent, name);
        parent->d_inode->i_op->lookup(dir, dentry);
        return dentry;
}
```

fs/romfs/inode.c

Ejemplos de sistemas de fichero son minix y romfs porque ellos son sencillos y pequeños. Por ejemplo, en fs/romfs/inode.c:

```
romfs_lookup(struct inode *dir, struct dentry *dentry) {
        const char *name = dentry->d_name.name;
        int len = dentry->d_name.len;
        char fsname[ROMFS_MAXFN];
        struct romfs_inode ri;
        unsigned long offset = dir->i_ino & ROMFH_MASK;
        for (;;) {
                romfs_copyfrom(dir, &ri, offset, ROMFH_SIZE);
                romfs_copyfrom(dir, fsname, offset+ROMFH_SIZE, len+1);
                if (strncmp (name, fsname, len) == 0)
                        break;
                /* next entry */
                offset = ntohl(ri.next) & ROMFH_MASK;
        }
        inode = iget(dir->i_sb, offset);
        d_add (dentry, inode);
        return 0;
}
```

romfs_copyfrom()

```
romfs_copyfrom(struct inode *i, void *dest,
                        unsigned long offset, unsigned long count) {
        struct buffer_head *bh;
        bh = bread(i->i_dev, offset>>ROMBSBITS, ROMBSIZE);
        memcpy(dest, ((char *)bh->b_data) + (offset & ROMBMASK), count);
        brelse(bh);
}
```