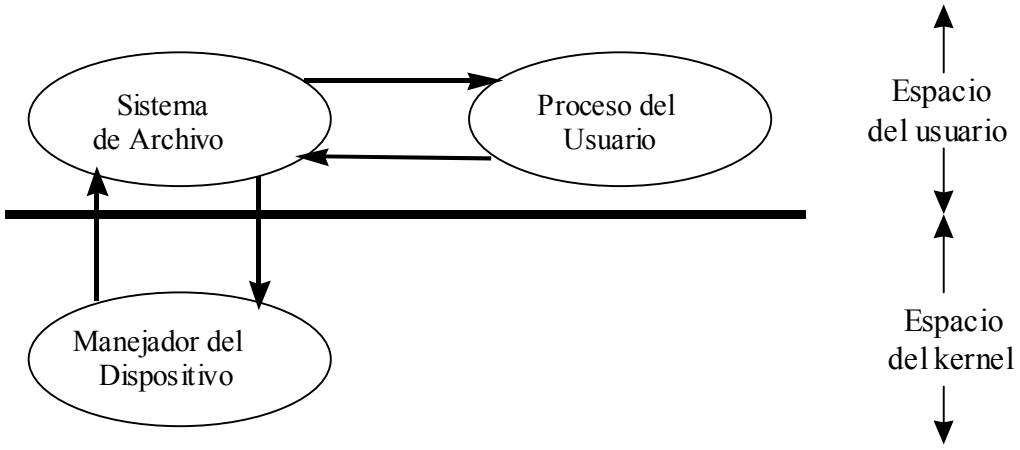
DEVICE.C

MINIX 2.0

INDICE

Introducción

En Minix para cada dispositivo de E/S existe un proceso encargado de la gestión del dispositivo (Manejadores de procesos). La interacción de los procesos de usuario con el manejador de dispositivo se realiza a través del sistema de archivo mediante el paso de mensajes. Minix introduce el concepto de **archivos especiales**, para lograr que los dispositivos de E/S sean tratados como archivos.



La interfaz del sistema de archivos con los manejadores se encuentra en el archivo **"device.c"**, el cuál contiene los procedimientos encargados de llevar a cabo esta comunicación, así como de servir de base para la implementación del **software de E/S independiente del dispositivo.** La función básica del software independiente del dispositivo consiste en ejecutar las funciones de E/S que son comunes a todos los dispositivos y proporcionar de esta forma una interfaz uniforme al software a nivel de usuario.

Archivos Especiales

Como ya mencionamos MINIX emplea el concepto de archivo especial para lograr que los dispositivos de E/S sean tratados como archivos, pudiéndose: leer o escribir con sus mismas llamadas al sistema; y controlar el acceso mediante los mismos bits RWX. Es decir para el MINIX un nombre de dispositivo , como */dev/tty0*, especifica exclusivamente el nodo i de un archivo especial.

Una diferencia es el tiempo de ejecución de una lectura o escritura; en archivos especiales se puede tardar horas en completarse la operación, como por ejemplo, un proceso esperando datos desde el teclado.

Como el FS verifica el estado del archivo especial, si no pudiera completarse la operación inmediatamente, se suspendería al proceso solicitante (absteniéndose de enviarle una contestación), hasta que otro proceso modificase este estado.

Otra diferencia es que los nodos i de los archivos especiales contienen dos números (dispositivo mayor y menor). El número de dispositivo mayor indica el tipo dispositivo (impresora, disco duro, etc.), y el menor especifica el dispositivo concreto (impresora 1, disco duro 3, etc.) que se va a usar. Se envía al manejador de E/S correspondiente del dispositivo mayor un mensaje con el número de dispositivo menor, la operación a realizar, número del proceso solicitante, dirección del buffer y número de bytes a transferir. Cuando el manejador complete la tarea, transfiere los datos al buffer de usuario bloqueado (en espera) y responde con un mensaje al FS, quien a su vez envía una respuesta al usuario para desbloquearlo.

Ejemplo del uso de un fichero especial (/dev/lp) usando las llamadas al sistema para ficheros convencionales:

```
main() {
  int fd;
  fd = open ("/dev/lp",2);
  write (fd,"hola",4);
  close (fd);
}
```

La entrada/salida en MINIX se realiza mediante el envío de mensajes a las tareas contenidas en el Kernel. La interfaz del sistema de archivos con estas tareas se encuentra en el archivo "device.c", el cual contiene procedimientos que realizan labores especiales para los archivos especiales.

Ficheros include y variables globales

```
#include "fs.h"
#include <fcntl.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include "dev.h"
#include "file.h"
#include "fproc.h"
#include "inode.h"
#include "param.h"

PRIVATE message dev_mess;
PRIVATE major, minor, task;

FORWARD _PROTOTYPE( void find_dev, (Dev_t dev));
```

Estructuras empleadas

- **"dmap":** es la estructura mas importante que maneja el device.c, a través de ella se realizan las llamadas a las funciones adecuadas para cada dispositivo. Contiene cuatro campos: punteros a las funciones para abrir, leer o escribir, y cerrar el disp., así como el identificador de la tarea del kernel encargada de ejecutar las operaciones anteriores. Esta tabla esta indexada por el número de disp. mayor.

```
EXTERN struct dmap {
dmap_t dmap_open;
dmap_t dmap_rw;
dmap_t dmap_close;
     int dmap_task;
} dmap[];
```

La tabla normalmente estará inicializada de esta manera:

dmap_open	dmap_rw	dmap_close	dmap_task
no_dev	no_dev	no_dev	0
dev_opcl	call_task	dev_opcl	MEM
dev_opcl	call_task	dev_opcl	FLOPPY
dev_opcl	call_task	dev_opcl	WINCHESTER
tty_open	call_task	dev_opcl	TTY
ctty_open	call_ctty	ctty_close	TTY
dev_opcl	call_task	dev_opcl	PRINTER

- **"inode"**: estructura referente a los nodos_i. (inode.c)
    - **i_dev**: dispositivo en el que esta el nodo_i.
    - **i_mode**: tipo de fichero, bits de protección, etc.
    - **i_zone[0]**: dirección del bloque de datos 0 (si el nodo_i corresponde a un archivo especial contiene la pareja de números dispositivo mayor-menor empaquetado.

- **"fproc"**: tabla de procesos del FS. (fproc.h)
    - **fs_tty**: dispositivos mayor/menor del controlador TTY.
    - **fp_filp**: tabla de descriptores de fichero de un proceso.
    - **fp_sesldr**: indica si el proceso es un "session leader", es decir, el primer proceso en abrir el terminal.

- **"filp"**: sirve de puente entre las tablas fproc e inode, y contiene una entrada por cada archivo abierto. (file.h)
    - **filp_ino**: puntero al nodo_i.
    - **filp_flags**: flags de open y fcntl.

Funciones

dev_io

Descripción

Realiza la lectura o escritura en un dispositivo de bloques o caracteres.

Procedimientos externos utilizados

- suspend: toma medidas para suspender el procesamiento de la llamada al sistema actual.

Parámetros de entrada

- op: operación a realizar (lectura, escritura, etc.).
- nonblock: indica si el disp. puede ser bloqueado.
- dev: número del dispositivo mayor-menor empaquetado.
- pos: posición en la que se va a realizar la operación.
- bytes: nº de bytes a transferir.
- proc: ¿a quién pertenece el espacio de direcciones en el que se encuentra el buffer?.
- buff: dirección virtual del buffer.

Parámetros de salida

Nº de bytes transferidos o resultado de la operación.
buff: datos leídos.

Algoritmo

1. Obtener el nº de dispositivo mayor, menor y la tarea.
2. Rellenar los campos del mensaje a enviar a la tarea que corresponda.
3. Llamar a la tarea a través de la tabla dmap.
4. Si la respuesta de la tarea nos indica que la operación no pudo completarse entonces
   5. Si la operación es OPEN entonces
      6. suspender al proceso
7. Devolver el estado

Código

```
PUBLIC int dev_io(op, nonblock, dev, pos, bytes, proc, buff)
int op;           /* DEV_READ, DEV_WRITE, DEV_IOCTL, etc. */
int nonblock;                 /* TRUE if nonblocking op */
dev_t dev;              /* major-minor device number */
off_t pos;              /* byte position */
int bytes;              /* how many bytes to transfer */
int proc;         /* in whose address space is buff? */
char *buff;             /* virtual address of the buffer */
{
/* Read or write from a device.  The parameter 'dev' tells which one. */

  find_dev(dev);        /* load the variables major, minor, and task */

  /* Set up the message passed to task. */
  dev_mess.m_type   = op;
  dev_mess.DEVICE   = (dev >> MINOR) & BYTE;
  dev_mess.POSITION = pos;
  dev_mess.PROC_NR  = proc;
```

```
  dev_mess.ADDRESS  = buff;
  dev_mess.COUNT    = bytes;
  dev_mess.TTY_FLAGS = nonblock; /* temporary kludge */

  /* Call the task. */
  (*dmap[major].dmap_rw)(task, &dev_mess);

  /* Task has completed.  See if call completed. */
  if (dev_mess.REP_STATUS == SUSPEND) {
   if (op == DEV_OPEN) task = XPOPEN;
   suspend(task);       /* suspend user */
  }

  return(dev_mess.REP_STATUS);
}
```

find_dev

Descripción

Extraer el nº de dispositivo mayor y menor del dato empaquetado, así como la tarea que les corresponde.

Parámetros de entrada

- dev: número del dispositivo mayor-menor empaquetado.

Algoritmo

1. Obtener los números de dispositivo mayor y menor
2. Si el nº mayor no es correcto entonces
   3. poner a cero ambos números
4. Extraer de la tabla dmap el nº de la tarea que atiende al dispositivo.

Código

```
PRIVATE void find_dev(dev)
dev_t dev;              /* device */
{
/* Extract the major and minor device number from the parameter. */

  major = (dev >> MAJOR) & BYTE;    /* major device number */
  minor = (dev >> MINOR) & BYTE;    /* minor device number */
  if (major >= max_major) {
   major = minor = 0;         /* will fail with ENODEV */
  }
  task = dmap[major].dmap_task;     /* which task services the device */
}
```

do_ioctl

Descripción

Realiza la llamada al sistema IOCTL.

Procedimientos externos utilizados

- get_filp: nos devuelve la entrada en la tabla filp para un descriptor de fichero dado.
- suspend: toma medidas para suspender el procesamiento de la llamada al sistema actual.

Parámetros de salida

- Resultado de la llamada.

Algoritmo

1. Obtener la entrada en la tabla filp.
2. Si el proceso no tiene ningún archivo abierto entonces
   3. devolver error
4. Obtener el puntero al nodo_i
5. Si el nodo_i no perteneces a un archivo especial entonces
   6. devolver error
7. Encontrar los números de disp. mayor, menor y la tarea
8. Rellenar los campos del mensaje con los valores apropiados
9. Llamar a la tarea a través de la tabla dmap
10. Si la respuesta de la tarea nos indica que la operación no pudo completarese entonces
    11. Si el disp. no podía bloquearse entonces
        12. enviarle un mensaje de cancelación a la tarea
        13. Si la llamada ha sido interrumpida entonces
            14. devolver que el disp. no esta disponible
    15. si no
        16. suspender al proceso
17. Devolver el estado

Código

```
PUBLIC int do_ioctl()
{
/* Perform the ioctl(ls_fd, request, argx) system call (uses m2 fmt). */

  struct filp *f;
  register struct inode *rip;
  dev_t dev;

  if ( (f = get_filp(ls_fd)) == NIL_FILP) return(err_code);
  rip = f->filp_ino;          /* get inode pointer */
  if ( (rip->i_mode & I_TYPE) != I_CHAR_SPECIAL
   && (rip->i_mode & I_TYPE) != I_BLOCK_SPECIAL) return(ENOTTY);
  dev = (dev_t) rip->i_zone[0];
  find_dev(dev);

  dev_mess= m;

  dev_mess.m_type  = DEV_IOCTL;
  dev_mess.PROC_NR = who;
  dev_mess.TTY_LINE = minor;
```

```
  /* Call the task. */
  (*dmap[major].dmap_rw)(task, &dev_mess);

  /* Task has completed.  See if call completed. */
  if (dev_mess.REP_STATUS == SUSPEND) {
   if (f->filp_flags & O_NONBLOCK) {
     /* Not supposed to block. */
     dev_mess.m_type = CANCEL;
     dev_mess.PROC_NR = who;
     dev_mess.TTY_LINE = minor;
     (*dmap[major].dmap_rw)(task, &dev_mess);
     if (dev_mess.REP_STATUS == EINTR) dev_mess.REP_STATUS = EAGAIN;
   } else {
     suspend(task);           /* User must be suspended. */
   }
  }
#if ENABLE_BINCOMPAT
  m1.TTY_SPEK = dev_mess.TTY_SPEK; /* erase and kill */
  m1.TTY_FLAGS = dev_mess.TTY_FLAGS;     /* flags */
#endif
  return(dev_mess.REP_STATUS);
}
```

dev_opcl

Descripción

Es llamada a través de la tabla dmap para abrir o cerrar ficheros especiales.

Procedimientos externos utilizados

- suspend: toma medidas para suspender el procesamiento de la llamada al sistema actual.

Parámetros de entrada

- task_nr: nº de la tarea según el dispositivo.
- mess_ptr: mensaje a enviar a la tarea.

Algoritmo

1. Almacenar la operación a realizar.
2. Indicar en el mensaje el dispositivo menor y el proceso llamador.
3. Llamar a la tarea a través de *call_task*.
4. Si la respuesta de la tarea nos indica que la operación no pudo completarse entonces
   5. Si la operación es OPEN entonces
      6. suspender al proceso

Código

```
PUBLIC void dev_opcl(task_nr, mess_ptr)
int task_nr;                  /* which task */
message *mess_ptr;            /* message pointer */
{
/* Called from the dmap struct in table.c on opens & closes of special
files.*/

  int op;

  op = mess_ptr->m_type;      /* save DEV_OPEN or DEV_CLOSE for later */
  mess_ptr->DEVICE = (mess_ptr->DEVICE >> MINOR) & BYTE;
  mess_ptr->PROC_NR = fp - fproc;

  call_task(task_nr, mess_ptr);

  /* Task has completed.  See if call completed. */
  if (mess_ptr->REP_STATUS == SUSPEND) {
   if (op == DEV_OPEN) task_nr = XPOPEN;
   suspend(task_nr);     /* suspend user */
  }
}
```

tty_open

Descripción

Es llamada a través de la tabla dmap para abrir terminales TTY.

Parámetros de entrada

- task_nr: nº de la tarea según el dispositivo.
- mess_ptr: mensaje a enviar a la tarea.

Algoritmo

1. Obtener el nº de dispositivo.
2. Rellenar los campos del mensaje a enviar a la tarea que corresponda.
3. Si el proceso no es un líder de sesión o no tiene un tty asociado entonces
   4. actualizar los flags
5. Si no
   6. buscar si existe un proceso con el mismo terminal asociado
   7. Si existe entonces
      8. actualizar los flags
9. Llamar a la función *dev_io*
10. Si se pudo abrir con éxito entonces
    11. asignar el terminal al proceso
12. Devolver el resultado de la llamada

Código

```
PUBLIC void tty_open(task_nr, mess_ptr)
int task_nr;
message *mess_ptr;
{
/* This procedure is called from the dmap struct in table.c on tty opens.
*/
  int r;  dev_t dev;  int flags, proc;  register struct fproc *rfp;
```

```
  dev = (dev_t) mess_ptr->DEVICE;
  flags = mess_ptr->COUNT;
  proc = fp - fproc;

  /* Add O_NOCTTY to the flags if this process is not a session leader, or
   * if it already has a controlling tty, or if it is someone elses
   * controlling tty.
   */
  if (!fp->fp_sesldr || fp->fp_tty != 0) {
  flags |= O_NOCTTY;
  } else {
  for (rfp = &fproc[LOW_USER]; rfp < &fproc[NR_PROCS]; rfp++) {
     if (rfp->fp_tty == dev) flags |= O_NOCTTY;
  }
  }

  r = dev_io(DEV_OPEN, mode, dev, (off_t) 0, flags, proc, NIL_PTR);

  if (r == 1) {
  fp->fp_tty = dev;
  r = OK;
  }

  mess_ptr->REP_STATUS = r;
}
```

ctty_open

Descripción

Es llamada a través de la tabla dmap para abrir el console.

Parámetros de entrada

- task_nr: nº de la tarea según el dispositivo.
- mess_ptr: mensaje a enviar a la tarea.

Algoritmo

1. Si el disp. tty esta inicializado entonces
   2. devolver OK
3. si no
   4. devolver que el dispositivo no existe

Código

```
PUBLIC void ctty_open(task_nr, mess_ptr)
int task_nr;
message *mess_ptr;
{
/* This procedure is called from the dmap struct in table.c on opening
 * /dev/tty, the magic device that translates to the controlling tty.
 */

  mess_ptr->REP_STATUS = fp->fp_tty == 0 ? ENXIO : OK;
}
```

ctty_close

Descripción

Es llamada a través de la tabla dmap para cerrar el console.

Parámetros de entrada

- task_nr: nº de la tarea según el dispositivo.
- mess_ptr: mensaje a enviar a la tarea.

Algoritmo

1. Devuelve en el campo de estado del mensaje el valor OK.

Código

```
PUBLIC void ctty_close(task_nr, mess_ptr)
int task_nr;
message *mess_ptr;
{
/* Close /dev/tty. */

  mess_ptr->REP_STATUS = OK;
}
```

do_setsid

Descripción

Realiza la llamada al sistema SETSID.

Parámetros de salida

- Resultado de la llamada.

Algoritmo

1. Si el proceso que realiza la llamada no es el MM entonces
   2. retornamos que la función de sistema no esta implementada
3. Hacer el proceso un “session leader” con ninguna tty asignada

Código

```
PUBLIC int do_setsid()
{
/* Perform the FS side of the SETSID call, i.e. get rid of the controlling
 * terminal of a process, and make the process a session leader.
 */
  register struct fproc *rfp;

  /* Only MM may do the SETSID call directly. */
  if (who != MM_PROC_NR) return(ENOSYS);

  /* Make the process a session leader with no controlling tty. */
  rfp = &fproc[slot1];
  rfp->fp_sesldr = TRUE;
  rfp->fp_tty = 0;
}
```

call_task

Descripción

Es el procedimiento que se comunica mediante mensajes con la tarea del kernel en función del dispositivo.

Procedimientos externos utilizados

- sendrec: enviar un mensaje y esperar por la respuesta.
- receive: esperar por la recepción de un mensaje.
- revive: marca un proceso suspendido como candidato a ejecución otra vez.
- panic: visualización de un mensaje de error.

Parámetros de entrada

- task_nr: nº de la tarea según el dispositivo.
- mess_ptr: mensaje a enviar a la tarea.

Parámetros de salida

- Resultado de la llamada sobre el mensaje.

Algoritmo

Mientras falle el envio de un mensaje (debido a que la tarea esta intentando enviar un mensaje para "revivir" una petición anterior) hacer
  recibir el mensaje
  Si el mensaje recibido no es OK entonces
    salir del bucle
  Si se trata de enviar un mensaje de cancelación a una tarea que acaba de indicar que ha terminado entonces
    ignorar la respuesta y la petición de cancelación
  Si el mensaje recibido no es de "revivir" entonces
    indicar que se ha recibido una respuesta extraña
    continuar en la siguiente iteración del bucle
  Revivir el proceso
Bucle infinito
  Si el mensaje recibido no es OK entonces
    PANIC, no se puede enviar o recibir
  Si el proceso al que se envió el mensaje obtuvo un resultado entonces
    salir del bucle
  Si el mensaje recibido no es de "revivir" entonces
    indicar que se ha recibido una respuesta extraña
    continuar en la siguiente iteración del bucle
  Revivir el proceso
  esperar por una respuesta al mensaje

Código

```
PUBLIC void call_task(task_nr, mess_ptr)
int task_nr;                  /* which task to call */
message *mess_ptr;            /* pointer to message for task */
{
/* All file system I/O ultimately comes down to I/O on major/minor device
 * pairs.  These lead to calls on the following routines via the dmap
table.
```

```
 */

  int r, proc_nr;
  message local_m;

  proc_nr = mess_ptr->PROC_NR;

  while ((r = sendrec(task_nr, mess_ptr)) == ELOCKED) {
  /* sendrec() failed to avoid deadlock. The task 'task_nr' is
   * trying to send a REVIVE message for an earlier request.
   * Handle it and go try again.
   */
  if ((r = receive(task_nr, &local_m)) != OK) break;

  /* If we're trying to send a cancel message to a task which has just
   * sent a completion reply, ignore the reply and abort the cancel
   * request. The caller will do the revive for the process.
   */
  if (mess_ptr->m_type == CANCEL && local_m.REP_PROC_NR == proc_nr)
    return;

  /* Otherwise it should be a REVIVE. */
  if (local_m.m_type != REVIVE) {
    printf(
    "fs: strange device reply from %d, type = %d, proc = %d\n",
      local_m.m_source,
      local_m.m_type, local_m.REP_PROC_NR);
    continue;
  }

  revive(local_m.REP_PROC_NR, local_m.REP_STATUS);
  }

  /* The message received may be a reply to this call, or a REVIVE for some
   * other process.
   */
  for (;;) {
  if (r != OK) panic("call_task: can't send/receive", NO_NUM);

  /* Did the process we did the sendrec() for get a result? */
  if (mess_ptr->REP_PROC_NR == proc_nr) break;

  /* Otherwise it should be a REVIVE. */
  if (mess_ptr->m_type != REVIVE) {
    printf(
    "fs: strange device reply from %d, type = %d, proc = %d\n",
      mess_ptr->m_source,
      mess_ptr->m_type, mess_ptr->REP_PROC_NR);
    continue;
  }
  revive(mess_ptr->REP_PROC_NR, mess_ptr->REP_STATUS);

  r = receive(task_nr, mess_ptr);
  }
}
```

call_ctty

Descripción

Se llama solo para el dispositivo /dev/tty y adecua el mensaje para su tratamiento.

Parámetros de entrada

- task_nr: nº de la tarea según el dispositivo (no usado, sirve de compatibilidad con dmap_t).
- mess_ptr: mensaje a enviar a la tarea.

Parámetros de salida

- Resultado de la llamada sobre el mensaje.

Algoritmo

1. Si el proceso no tiene un controlador de tty asignado entonces
   2. devolver error de E/S
3. Obtenemos la tarea, los números mayor y menor adecuados para este dispositivo concreto
4. Llamar a call_task

Código

```
PUBLIC void call_ctty(task_nr, mess_ptr)
int task_nr;                  /* not used - for compatibility with dmap_t
*/
message *mess_ptr;            /* pointer to message for task */
{
/* This routine is only called for one device, namely /dev/tty.  Its job
 * is to change the message to use the controlling terminal, instead of the
 * major/minor pair for /dev/tty itself.
 */

  int major_device;

  if (fp->fp_tty == 0) {
   /* No controlling tty present anymore, return an I/O error. */
   mess_ptr->REP_STATUS = EIO;
   return;
  }
  major_device = (fp->fp_tty >> MAJOR) & BYTE;
  task_nr = dmap[major_device].dmap_task; /* task for controlling tty */
  mess_ptr->DEVICE = (fp->fp_tty >> MINOR) & BYTE;
  call_task(task_nr, mess_ptr);
}
```

no_dev

Descripción

Procedimiento tonto para indicar que no hay un dispositivo asociado.

Parámetros de entrada

- task_nr: nº de la tarea según el dispositivo (no usado, sirve de compatibilidad con dmap_t).
- mess_ptr: mensaje a enviar a la tarea.

Parámetros de salida

Constante que indica que no hay un dispositivo.

Algoritmo

1. Devolver la constante ENODEV sobre el campo de estado del mensaje

Código

```
PUBLIC void no_dev(task_nr, m_ptr)
int task_nr;                    /* not used - for compatibility with dmap_t
*/
message *m_ptr;                 /* message pointer */
{
/* No device there. */

  m_ptr->REP_STATUS = ENODEV;
}
```

Cuestiones

1. ¿Qué ventaja aporta a los programadores el concepto que introduce MINIX de los llamados archivos especiales?

2. ¿Cuál es la utilidad de la tabla DMAP?

3. ¿Cómo se obtienen el par de números de dispositivos mayor y menor?

Ejemplo de algunas de las llamadas del sistema