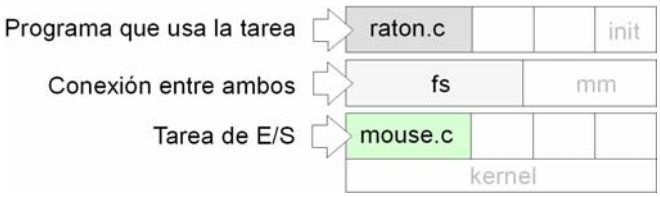
Práctica 3: Creación de servicios en un manejador de dispositivo

El objeto de esta práctica es dotar de servicios a un manejador de dispositivo y utilizarlos desde un programa de usuario mediante primitivas de gestión de ficheros. El guión de práctica es el que sigue:

1. Introducción.



La figura muestra el trabajo que vamos a hacer en la práctica. Ya hemos generado una imagen de Minix que contiene `mouse`, un manejador de dispositivo que existe pero que no presta servicios. En esta práctica se trata de dotar a mouse de dos servicios, abrirlo y cerrarlo. Hecho esto, ¿cómo se accede a los servicios de un manejador? Los programas de usuario en sistemas POSIX realizan operaciones sobre los manejadores mediante las funciones `open`, `close`, `ioctl`, `read` y `write`. Estas operaciones se realizan sobre ficheros especiales, contenidos en el directorio `/dev`. Crearemos en el sistema el dispositivo `/dev/mouse` y a continuación lo abriremos y cerraremos en un programa de usuario que vamos a construir al efecto, `raton.c`.

2. Creación del dispositivo en el sistema.

Antes de operar sobre un dispositivo es preciso darle de alta en el sistema. Un dispositivo se identifica mediante un par de enteros. El primero se denomina el número principal y es el código del manejador de dispositivo que lo atiende. El segundo es el número secundario, y es el número de orden entre los dispositivos que atiende el manejador. El siguiente mandato crea el dispositivo de caracteres mouse con número principal 15 y número secundario 0.

```
! su
# mknod /dev/mouse c 15 0
```

3. Registro del dispositivo en el sistema de ficheros.

¿Dado el número principal, cómo el sistema de ficheros determina la tarea a quien enviar el mensaje? Esta relación viene dada por la tabla `dmap`, en `/usr/src/fs/table.c`, que se muestra a continuación. Esta tabla tiene tantas entradas como tareas compiladas en el núcleo. Cada entrada de `dmap` es una estructura de tipo `dmap`, que contiene tres punteros a función y un entero que es el número de proceso correspondiente a la tarea. La tabla está ordenada según el número principal del dispositivo al que sirve la tarea. Así, si deseamos conocer cuál es el numero de la tarea que sirve al dispositivo con número principal 2 (el disco flexible) lo hallaremos en el cuarto componente de la entrada número 2 de la tabla, `FLOPPY`, definido en `/usr/include/minix/com.h`. El segundo componente de la estructura `dmap` es la que el sistema de ficheros utiliza para enviar el mensaje a la tarea correspondiente. En casi todos los casos es `call_task`.

```
PUBLIC struct dmap dmap[] = {
/*  ?   Open       Read/Write   Close       Task #      Device  File
    -   ----       ----------   -----       -------     ------  ----       */
  DT(1, no_dev,    no_dev,      no_dev,     0)           /* 0 = not used   */
  DT(1, dev_opcl,  call_task,   dev_opcl,   MEM)         /* 1 = /dev/mem   */
  DT(1, dev_opcl,  call_task,   dev_opcl,   FLOPPY)      /* 2 = /dev/fd0   */
  DT(ENABLE_WINI,
        dev_opcl,  call_task,   dev_opcl,   WINCHESTER)  /* 3 = /dev/hd0   */
  DT(1, tty_open,  call_task,   dev_opcl,   TTY)         /* 4 = /dev/tty00 */
  DT(1, ctty_open, call_ctty,   ctty_close, TTY)         /* 5 = /dev/tty   */
  DT(1, dev_opcl,  call_task,   dev_opcl,   PRINTER)     /* 6 = /dev/lp    */

  DT(ENABLE_NETWORKING,
        net_open,  call_task,   dev_opcl,   INET_PROC_NR)/* 7 = /dev/ip    */
  DT(ENABLE_CDROM,
        dev_opcl,  call_task,   dev_opcl,   CDROM)       /* 8 = /dev/cd0   */
  DT(0, 0,         0,           0,          0)           /* 9 = not used   */
  DT(ENABLE_SCSI,
        dev_opcl,  call_task,   dev_opcl,   SCSI)        /*10 = /dev/sd0   */
  DT(0, 0,         0,           0,          0)           /*11 = not used   */
  DT(0, 0,         0,           0,          0)           /*12 = not used   */
  DT(ENABLE_AUDIO,
        dev_opcl,  call_task,   dev_opcl,   AUDIO)       /*13 = /dev/audio */
  DT(ENABLE_AUDIO,
        dev_opcl,  call_task,   dev_opcl,   MIXER)       /*14 = /dev/mixer */
};
```

Modificación 1.
Hemos creado el dispositivo `/dev/mouse` con número principal 15. En la tabla `dmap`, por lo tanto, hay que añadir al final la entrada número 15. Para ello hay modificar el fichero table.c y recompilar el sistema de ficheros.

```
! cd /usr/src/fs
! mined table.c
! make
```

4. Extendiendo el manejador de dispositivo.

Modificación 2.
Se trata de la segunda versión de `/usr/src/kernel/mouse.c`. Esta vez el sistema de ficheros la envía mensajes y es preciso replicarle mediante `send`.

```
#include "kernel.h"
#include <minix/com.h>
#include <minix/callnr.h>
void mouse_task()
{
  message mess;
  int     status;
  while(1) {
    receive(ANY, &mess);
    switch(mess.m_type) {
      case DEV_OPEN:
        printf("...");
        break;
      case DEV_CLOSE:
        printf("...");
        break;
    }
    mess.m_type      = TASK_REPLY;
    mess.REP_PROC_NR = mess.PROC_NR;
    mess.REP_STATUS  = OK;
    if(OK != (status = send(mess.m_source, &mess)))
      panic("Error en mouse\n", status);
  }
};
```

5. **Escribiendo el programa de usuario**.

Finalmente, escribiremos el programa de usuario `raton.c`, que prueba los nuevos servicios.

```
! cd /usr
! mkdir home
! cd home
! mined raton.c
```

```
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
main()
{
  int fd;
  if(0 > (fd = open("/dev/mouse", 0)) {
    perror("Raton, open");
    exit(1);
  }
  printf("Raton abierto\n");
  if(0 > close(fd)) {
    perror("Raton, close");
    exit(1);
  }
  printf("Practica superada con exito\n");
}
```

```
! cc raton.c -o raton
! raton
```