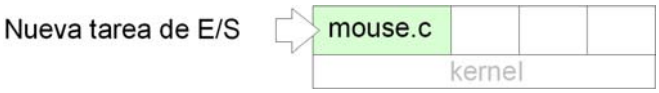
Práctica 2: Extendiendo Minix con un nuevo manejador de dispositivo

El objeto de esta práctica es aprender a añadir un nuevo manejador de dispositivo al sistema operativo. El guión de práctica es el que sigue:

1. Introducción.



La figura muestra el trabajo que vamos a hacer en la práctica. En primer lugar generaremos una imagen de Minix que contenga un nuevo manejador de dispositivo. En Minix los manejadores son conocidos como tareas de E/S o simplemente tareas. En esta práctica no se trata de escribir la tarea completa con su acceso al hardware, etc., sino la tarea más sencilla posible. Su funcionalidad se irá añadiendo de forma progresiva en prácticas sucesivas.

2. Registro de la tarea en el núcleo.

Modificación 1.
En clase de teoría vimos los números de proceso. Las tareas tienen números negativos según establece el fichero `/usr/include/MINIX/com.h`. Las ocho tareas inferiores son obligatorias. No así otras como la tarea ethernet. Observemos el papel que juegan las constantes `ENABLE_xx`. Hay una por cada tarea. Si una de estas constantes está definida como 0, la tarea no se incorpora al núcleo. Sí se incorpora si vale 1. Observemos que `TTY` es la más negativa. Esto no es un capricho. La razón es que así se puede invocar `printf` en todas las tareas.

```
...
#define TTY          (DL_ETH - 1)
#define DL_ETH       (CDROM - ENABLE_NETWORKING)
#define CDROM        (AUDIO - ENABLE_CDROM)
#define AUDIO        (MIXER - ENABLE_AUDIO)
#define MIXER        (SCSI - ENABLE_AUDIO)
#define SCSI         (WINCHESTER - ENABLE_SCSI)
#define WINCHESTER   (SYN_ALRM_TASK - ENABLE_WINI)
                              /* winchester (hard) disk class */
#define SYN_ALRM_TASK  -8     /* task to send CLOCK_INT messages */
#define IDLE           -7     /* task to run when there's nothing to run */
#define PRINTER        -6     /* printer I/O class */
#define FLOPPY         -5     /* floppy disk class */
#define MEM            -4     /* /dev/ram, /dev/(k)mem and /dev/null class */
#define CLOCK          -3     /* clock class */
#define SYSTASK        -2     /* internal functions */
#define HARDWARE       -1     /* used as source on interrupt generated msgs */
...
```

La primera modificación, pues, consiste en añadir la línea correspondiente a la nueva tarea:

```
#define MOUSE ...
```

Modificación 2.
Las constantes `ENABLE_xx` se definen en el fichero `/usr/include/minix/config.h`. Aquí hay que definir `ENABLE_MOUSE`.

Modificación 3.
En el fichero `/usr/include/minix/const.h` hay que modificar la constante del número total de tareas para tener en cuenta `ENABLE_MOUSE`.

```
#define NR_TASKS (9 + ENABLE_WINI + ENABLE_SCSI + ENABLE_CDROM \
                    + ENABLE_NETWORKING + 2 * ENABLE_AUDIO)
```

Modificación 4.
En el fichero `/usr/src/kernel/table.c` cada entrada del vector `tasktab` contiene el punto de entrada de la tarea correspondiente, el tamaño de su pila y una cadena de caracteres que la identifica:

```
struct tasktab tasktab[] = {
        { tty_task,             TTY_STACK,      "TTY"         },
        ...
        { syn_alrm_task,        SYN_ALRM_STACK, "SYN_AL"      },
        { idle_task,            IDLE_STACK,     "IDLE"        },
        { printer_task,         PRINTER_STACK,  "PRINTER"     },
        { floppy_task,          FLOP_STACK,     "FLOPPY"      },
        { mem_task,             MEM_STACK,      "MEMORY"      },
        { clock_task,           CLOCK_STACK,    "CLOCK"       },
        { sys_task,             SYS_STACK,      "SYS"         },
        { 0,                    HARDWARE_STACK, "HARDWAR"     },
        { 0,                    0,              "MM"          },
        { 0,                    0,              "FS"          },
        { 0,                    0,              "INIT"        },
};
```

Hay que añadir una entrada nueva para la tarea del ratón. ¡Atención aquí! No se puede añadir la entrada en cualquier sitio. Hay que hacerlo en concordancia con el número de tarea que ha tomado en `/usr/include/MINIX/com.h`.

Modificación 5.
En el fichero `/usr/src/kernel/proto.h` hay que declarar el punto de entrada de la nueva tarea, tal vez `mouse_task`.

3. Escribiendo la tarea.

Las modificaciones anteriores registran la nueva tarea en el núcleo. Cuando Minix arranque será planificada como cualquier otra. Evidentemente la tarea hay que escribirla. A continuación se muestra el esqueleto de una tarea de entrada-salida. Es prácticamente igual para todas.

```
tarea()
{
  mensaje mens;
  int     r, emisor;

  inicializar();
  while(TRUE) {
      receive(ANY, &mens);
      emisor = mens.fuente;
      switch(mens.tipo)
      {
        case READ:     r = do_read();     break;
        case WRITE:    r = do_write();    break;
        case INTERRUP: r = do_interrup(); break;
        case OTRO:     r = do_otro();     break;
        default:       r = ERROR;
      }
      mens.tipo       = TASK_REPLAY;
      mens.REP_STATUS = r;
      send(emisor, &mens);
  }
}
```

Modificación 6.
Crearemos el fichero `/usr/src/kernel/mouse.c` como sigue:

```
#include "kernel.h"
#include <minix/com.h>
void mouse_task()
{
  message mess;
```

```
    printf("Soy el raton\n");
    receive(ANY, &mess);
  }
```

Esta es la tarea más simple posible. Toma el control, se identifica y se bloquea en `receive`. De momento no vamos a enviarla ningún mensaje. Si así lo hiciésemos el resultado sería catastrófico, ya que `mouse_task` no ha sido invocada por nadie y, por lo tanto, no puede retornar. Obsérvese que el punto de entrada, `mouse_task`, debe haber sido registrado como una entrada de `table.c`.

4. Compilado el núcleo extendido.

Hay que entender que el programa anterior no es un programa de usuario autónomo que se pueda compilar con el mandato `cc`. Es una sección más de código de kernel y, como tal, hay que incluirlo en el `Makefile` `/usr/src/kernel/Makefile`.

Modificación 7.
Editar `Makefile` con `mined`. Hay que

1. Añadir `mouse.o` a la variable `OBJS`
2. Añadir el objetivo `mouse.o: mouse.h`

La primera medida hará que el mandato `make` busque y compile mouse.c. a `mouse.o`. La segunda hará que cualquier modificación de `mouse.h` obligue a recompilar `mouse.c`.

A continuación debemos generar la nueva imagen y arrancarla tal y como hicimos en la práctica 1.