MANUAL MINIKERNEL
AMPLIACIÓN DE SISTEMAS OPERATIVOS

Descripción del “minikernel”

El entorno de desarrollo de la práctica intenta imitar dentro de lo que cabe el comportamiento y estructura de un sistema operativo real. En una primera aproximación, en este entorno se pueden diferenciar tres componentes principales (que se corresponden con los tres subdirectorios que presenta la jerarquía de ficheros del entorno):

- El programa cargador del sistema operativo (directorio “boot”).
- El entorno propiamente dicho (directorio “minikernel”), que incluye tanto el *hardware virtual* (módulo “HAL”, *Hardware Abstraction Layer*) como el sistema operativo (módulo “kernel”).
- Los programas de usuario (directorio “usuario”).

A continuación, se describe cada una de estas partes.

Carga del sistema operativo

De forma similar a lo que ocurre en un sistema real, en este entorno existe un programa de arranque que se encarga de cargar el sistema operativo en memoria y pasar el control a su punto de entrada inicial. Este procedimiento imita el modo de arranque de los sistemas operativos reales que se realiza desde un programa cargador. El programa cargador se encuentra en el subdirectorio “boot” y, en un alarde de originalidad, se denomina “boot”. Para arrancar el sistema operativo y con ello el entorno de la práctica, se debe ejecutar dicho programa pasándole como argumento el nombre del fichero que contiene el sistema operativo. Así, si se está situado en el directorio base de la práctica, se podría usar el siguiente mandato:

```
boot/boot minikernel/kernel
```

Una vez arrancado, el sistema operativo continuará ejecutando mientras haya procesos de usuario que ejecutar. Nótese que este comportamiento es diferente al de un sistema real, donde el administrador tiene que realizar alguna operación explícita para parar la ejecución del sistema operativo. Sin embargo, este modelo de ejecución resulta más conveniente a la hora de probar la práctica.

El módulo HAL

El objetivo principal de este módulo es implementar la capa del hardware y ofrecer servicios que permitan al sistema operativo su manejo. Las principales características de este “hardware” son las siguientes:

- El “procesador” tiene los dos modos de ejecución clásicos: modo privilegiado o núcleo, en el que se ejecuta código del sistema operativo, y modo usuario, que corresponde con la ejecución del código de procesos de usuario.
- El procesador sólo pasa de modo usuario a privilegiado debido a la ocurrencia de algún tipo de interrupción.
- El registro de estado del procesador almacena el modo de ejecución del mismo. Dado que en el retorno del tratamiento de una interrupción, sea del tipo que sea, se restaura el registro de estado, el paso de modo privilegiado a modo usuario se producirá en esta situación de retorno de interrupción cuando en el registro de estado que se restaura el modo es usuario.
- En el sistema hay dos dispositivos de entrada/salida basados en interrupciones: el terminal y el reloj.
- El terminal es de tipo proyectado en memoria. Como se verá en el capítulo de entrada/salida, esto significa que, realmente, el terminal está formado por dos dispositivos independientes: la pantalla y el teclado. La salida de datos a la pantalla del terminal se realiza escribiendo directamente en su memoria de vídeo, no implicando el uso de interrupciones. La entrada de datos mediante el teclado, sin embargo, está dirigida por las interrupciones que se producen cada vez que se pulsa una tecla.
- El reloj temporizador tiene una frecuencia de interrupción programable. Además de este temporizador que interrumpe periódicamente, hay un reloj alimentado con una batería donde se mantiene la hora mientras el equipo está apagado. Este reloj mantiene la hora como el número de milisegundos transcurridos desde el 1 de enero de 1970. Habitualmente, a este reloj se le denomina reloj CMOS.
- El tratamiento de las interrupciones se realiza mediante el uso de una tabla de vectores de interrupción. Hay seis vectores disponibles que corresponden con:
  - Vector 0: Excepción aritmética.
  - Vector 1: Excepción por acceso a memoria inválido.
  - Vector 2: Interrupción de reloj.
  - Vector 3: Interrupción del terminal.
  - Vector 4: Llamada al sistema.
  - Vector 5: Interrupción software.
- Se trata de un procesador con múltiples niveles de interrupción que, de mayor a menor prioridad, son los siguientes:
  - Nivel 3: Interrupción de reloj.
  - Nivel 2: Interrupción del terminal.
  - Nivel 1: Interrupción software y llamada al sistema.
  - Nivel 0: Ejecución en modo usuario.

- En cada momento el procesador ejecuta en un determinado nivel de interrupción, cuyo valor está almacenado en el registro de estado, y sólo admite interrupciones de un nivel superior al actual.
- Inicialmente, el procesador ejecuta en modo privilegiado y en el nivel de interrupción máximo, por lo que todas las interrupciones están inhibidas.
- Cuando el procesador ejecuta en modo usuario (código de los procesos de usuario), ejecuta con un nivel 0, lo que hace que estén habilitadas todas las interrupciones.
- Cuando se produce una interrupción, sea del tipo que sea, el procesador realiza el tratamiento habitual, esto es, almacenar el contador de programa y el registro de estado en la pila, poner el procesador en modo privilegiado, fijar el nivel de interrupción de acuerdo con el tipo de interrupción recibida y cargar en el contador de programa el valor almacenado en el vector correspondiente. Evidentemente, al tratarse de operaciones realizadas por el hardware, todas estas operaciones no son visibles al programador del sistema operativo.
- La finalización de una rutina de interrupción conlleva la ejecución de una instrucción de retorno de interrupción que restaurará el nivel de interrupción y el modo de ejecución previos.
- Se dispone de una instrucción que permite modificar explícitamente el nivel de interrupción del procesador.
- Se trata de un procesador con mapas de entrada/salida y memoria separados. Por tanto, hay que usar instrucciones específicas de entrada/salida para acceder a los puertos de los dispositivos. Hay que aclarar que, realmente, sólo hay un puerto de entrada/salida que corresponde con el registro de datos del teclado, el cual, cuando se produce una interrupción, contiene la tecla pulsada.
- El procesador dispone de seis registros de propósito general de 32 bits, que sólo se tendrán que usar explícitamente para el paso de parámetros en las llamadas al sistema.
- La interrupción software es de tipo no expulsivo y se usará únicamente para la planificación de procesos, concretamente, para la realización de cambios de contexto involuntarios. Téngase en cuenta que se pretende construir un núcleo no expulsivo. Asimismo, obsérvese que, aunque la interrupción software de planificación sea una interrupción software de proceso, al tratarse de un sistema monoprocesador, siempre irá dirigida al proceso en ejecución, por lo que no es necesario especificar el destino de la misma.

Además de incluir funcionalidad relacionada con el hardware, este módulo también proporciona funciones de más alto nivel vinculadas con la gestión de memoria. Con ello, se pretende que el alumno se centre en los aspectos relacionados con la gestión de procesos y la entrada/salida, y no en los aspectos relacionados con la gestión de memoria.

Las funciones ofrecidas por este módulo se pueden clasificar en las siguientes categorías:

- Operaciones vinculadas a la iniciación de los controladores de dispositivos. En su arranque el sistema operativo debe asegurarse de que los controladores de los dispositivos se inician con un estado adecuado. El

módulo HAL ofrece una función de este tipo por cada uno de los dos dispositivos existentes.

- Iniciación del controlador de teclado.

```
void iniciar_cont_teclado();
```

- Iniciación del controlador de reloj. Se especifica como parámetro cuál es la frecuencia de interrupción deseada (número de interrupciones de reloj por segundo).

```
void iniciar_cont_reloj(int ticks_por_seg);
```

- Operaciones relacionadas con las interrupciones. En su fase de arranque, el sistema operativo debe iniciar el controlador de interrupciones a un estado válido y deberá instalar en la tabla de manejadores sus rutinas de tratamiento para cada uno de los vectores que hay en el sistema. Asimismo, se proporciona un servicio para cambiar explícitamente el nivel de interrupción del procesador y otro para activar una interrupción software.
  - Iniciación del controlador de interrupciones.

```
void iniciar_cont_int();
```

- Instalación de un manejador de interrupciones. Instala la función manejadora manej en el vector correspondiente a nvector. Existen varias constantes que facilitan la especificación del número de vector.

```
#define EXC_ARITM 0       /* excepción aritmética */
#define EXC_MEM 1         /* excepción en acceso a memoria */
#define INT_RELOJ 2       /* interrupción de reloj */
#define INT_TERMINAL 3    /* interrupción de terminal */
#define LLAM_SIS 4        /* vector usado para llamadas */
#define INT_SW 5/* vector usado para int. soft. */
void instal_man_int(int nvector, void (*manej)());
```

- Esta función fija el nivel de interrupción del procesador devolviendo el previo. Permite establecer explícitamente un determinado nivel de interrupción, lo que habilita las interrupciones con un nivel superior e inhabilita las que tienen un nivel igual o inferior. Devuelve el nivel de interrupción anterior para así, si se considera oportuno, poder restaurarlo posteriormente usando esta misma función. Están definidas tres constantes que representan los tres niveles de interrupción del sistema.

```
#define NIVEL_1 1 /* Int. Software */
#define NIVEL_2 2 /* Int. Terminal */
#define NIVEL_3 3 /* Int. Reloj */
int fijar_nivel_int(int nivel);
```

- Esta función provoca la activación de una interrupción software, que será tratada cuando el nivel de interrupción del procesador lo posibilite. Nótese que

no se proporciona ninguna función para desactivar la interrupción software una vez activada.

```
void activar_int_SW();
```

- Esta función consulta el registro de estado salvado por la interrupción actual y permite conocer si previamente se estaba ejecutando en modo usuario, devolviendo un valor verdadero en tal caso.

```
int viene_de_modo_usuario();
```

- Operaciones de gestión de la información de contexto del proceso. En el contexto del proceso (tipo “contexto_t”), se almacena una copia de los registros del procesador con los valores correspondientes a la última vez que ejecutó este proceso. Se ofrecen funciones para crear el contexto inicial de un nuevo proceso, así como para realizar un cambio de contexto, o sea, salvar el contexto de un proceso y restaurar el de otro.
- Este servicio crea el contexto inicial del proceso estableciendo los valores iniciales de los registros contador de programa (parámetro “pc_inicial”) y puntero de pila, a partir de la dirección inicial de la pila (parámetro “inicio_pila”) y su tamaño (parámetro “tam_pila”). Además, recibe como parámetro una referencia al mapa de memoria del proceso (parámetro “memoria”), que se debe haber creado previamente. Esta función devuelve un contexto iniciado de acuerdo con los valores especificados (parámetro de salida “contexto_ini”). Es importante resaltar que la copia del registro de estado dentro del contexto se inicia con un nivel de interrupción 0 y con un modo de ejecución usuario. De esta forma, cuando el sistema operativo active el proceso por primera vez mediante un cambio de contexto, el código del proceso se ejecutará en el modo y nivel de interrupción adecuados.

```
void fijar_contexto_ini(
   void *memoria,
   void *inicio_pila,
   int tam_pila,
   void *pc_inicial,
   contexto_t *contexto_ini
);
```

- Esta rutina salva el contexto de un proceso y restaura el de otro. Concretamente, la salvaguarda consiste en copiar el estado actual de los registros del procesador en el parámetro de salida “contexto_a_salvar”. Por su parte, la restauración implica copiar el contexto recibido en el parámetro de entrada “contexto_a_restaurar” en los registros del procesador. Nótese que, al terminar la operación de restauración, se ha “congelado” la ejecución del proceso que invocó esta rutina y se ha “descongelado” la ejecución del proceso restaurado justo por donde se quedó la última vez que ejecutó, ya sea en otra llamada a “cambio_contexto”, si ya ha ejecutado previamente, o desde su contexto inicial, si ésta es la primera vez que ejecuta. El proceso “congelado” no volverá a ejecutar, y, por tanto, no retornará de la llamada a

la función de “cambio_contexto”, hasta que otro proceso llame a esta misma rutina especificando como contexto a restaurar el de este proceso. Hay que resaltar que, dado que también se salva y restaura el registro de estado, se recuperará el nivel de interrupción que tenía previamente el proceso restaurado. Si no se especifica el proceso cuyo contexto debe salvarse (primer parámetro nulo), sólo se realiza la restauración.

```
void cambio_contexto(
   contexto_t *contexto_a_salvar,
   contexto_t *contexto_a_restaurar
);
```

- Funciones relacionadas con el mapa de memoria del proceso. Como se ha explicado previamente, el módulo HAL, además de las funciones vinculadas directamente con el hardware, incluye operaciones de alto nivel que gestionan todos los aspectos relacionados con la gestión de memoria. Se ofrecen servicios que permiten realizar operaciones tales como crear el mapa de memoria del proceso a partir del ejecutable y liberarla cuando sea oportuno, así como para crear la pila del proceso y liberarla.
  - Esta rutina crea el mapa de memoria a partir del ejecutable especificado (parámetro “prog”). Para ello, debe procesar el archivo ejecutable y crear las regiones de memoria (código y datos) correspondientes. Devuelve un identificador del mapa de memoria creado, así como la dirección de inicio del programa en el parámetro de salida “dir_ini”.

```
void *crear_imagen(char *prog, void **dir_ini);
```

  - Este servicio libera una imagen de memoria previamente creada (parámetro “mem”).

```
void liberar_imagen(void *mem);
```

  - Esta rutina reserva una zona de memoria para la región de pila. Se especifica como parámetro el tamaño de la misma, devolviendo como resultado la dirección inicial de la zona reservada.

```
void *crear_pila(int tam);
```

  - Este servicio libera una pila previamente creada (parámetro “pila”).

```
void liberar_pila(void *pila);
```

  - Operaciones misceláneas. En este apartado se agrupan una serie de funciones de utilidad diversa.
    - Rutinas que permiten leer y escribir, respectivamente, en los registros de propósito general del procesador.

```
long leer_registro(int nreg);
```

```
int escribir_registro(int nreg, long valor);
```

- Esta función lee y devuelve un byte del puerto de entrada/salida especificado (parámetro “dir_puerto”). El único puerto disponible en el sistema corresponde con el terminal.

```
#define DIR_TERMINAL 1
char leer_puerto(int dir_puerto);
```

- Ejecuta la instrucción “HALT” del procesador que detiene su ejecución hasta que se active una interrupción.

```
void halt();
```

- Esta función permite escribir en la pantalla los datos especificados en el parámetro “buffer” y cuyo tamaño corresponde con el parámetro “longi”. Para ello, copia en la memoria de vídeo del terminal dichos datos. La rutina de conveniencia “printk” se apoya en la anterior y permite escribir datos con formato, al estilo del clásico “printf” de C.

```
void escribir_ker(char *buffer, unsigned int longi);
int printk(const char *, ...);
```

- Esta función escribe el mensaje especificado (parámetro “mens”) por la pantalla y termina la ejecución del sistema operativo.

```
void panico(char *mens);
```

El módulo "kernel"

Este es el módulo que contiene la funcionalidad del sistema operativo. El alumno recibe como material de apoyo para la realización de la práctica una versión de este módulo que incluye una funcionalidad básica, que deberá modificarse siguiendo las pautas que se detallan posteriormente. A continuación, se describen las principales características generales de esta versión inicial:

- Iniciación. Una vez cargado el sistema operativo, el programa cargador pasa control al punto de entrada del mismo (en este caso, a la función “main” de este módulo). En este momento, el sistema inicia sus estructuras de datos, los dispositivos hardware e instala sus manejadores en la tabla de vectores. En último lugar, crea el proceso inicial “init” y lo activa pasándole el control. Nótese que durante esta fase el procesador ejecuta en modo privilegiado y las interrupciones están inhibidas (nivel de interrupción 3). Sin embargo, cuando se activa el proceso “init” restaurándose su contexto inicial, el procesador pasa a ejecutar en modo usuario y se habilitan automáticamente todas las interrupciones (nivel 0), puesto que en dicho contexto inicial se ha establecido previamente que esto sea así. Obsérvese que, una vez invocada la rutina de cambio de

contexto, no se puede volver nunca a esta función ya que no se ha salvado el contexto del flujo actual de ejecución. A partir de ese momento, el sistema operativo sólo se ejecutará cuando se produzca una llamada al sistema, una excepción o una interrupción de un dispositivo.

- Tratamiento de interrupciones externas. Las únicas fuentes externas de interrupciones son el reloj y el terminal. Las rutinas de tratamiento instaladas únicamente muestran un mensaje por la pantalla indicando la ocurrencia del evento. En el caso de la interrupción del teclado, la rutina además usa la función “leer_puerto” para obtener el carácter tecleado. En estas rutinas habrá que incluir progresivamente la funcionalidad pedida en las distintas prácticas.
- Tratamiento de interrupción software. Como ocurre con las interrupciones externas, la rutina de tratamiento sólo muestra un mensaje por la pantalla.
- Tratamiento de excepciones. Las dos posibles excepciones presentes en el sistema tienen un tratamiento común, que depende de en qué modo ejecutaba el procesador antes de producirse la excepción. Si estaba en modo usuario, se termina la ejecución del proceso actual. En caso contrario, se trata de un error del propio sistema operativo. Por tanto, se invoca la rutina “panico” para terminar su ejecución.
- Llamadas al sistema. Existe una única rutina de interrupción para todas las llamadas (rutina “tratar_llamsis”). Tanto el código numérico de la llamada como sus parámetros se pasan mediante registros. Por convención, el código se pasa en el registro 0 y los parámetros en los siguientes registros (parámetro 1 en registro 1, parámetro 2 en registro 2, y así sucesivamente hasta 5 parámetros). Asimismo, el resultado de la llamada se devuelve en el registro 0. La rutina “tratar_llamsis” obtiene el código numérico de la llamada e invoca indirectamente a través de “tabla_servicios” a la función correspondiente. Esta tabla guarda en cada posición la dirección de la rutina del sistema operativo que lleva a cabo la llamada al sistema cuyo código corresponde con dicha posición.
- En la versión inicial sólo hay tres llamadas disponibles. La función asociada con cada una de ellas está indicada en la posición correspondiente de “tabla_servicios” y, como se ha comentado previamente, será invocada desde “tratar_llamsis” cuando el valor recibido en el registro 0 así lo indique. Las llamadas disponibles en esta versión inicial son las siguientes:
  - Crear proceso. Crea un proceso que ejecuta el programa almacenado en el archivo especificado como parámetro. Esta llamada devolverá un -1 si hay un error y un 0 en caso contrario. Obsérvese que, en este caso, el código que realiza el tratamiento real de la llamada no se ha incluido en la propia función, sino que se ha delegado a una función auxiliar. Puesto que esta rutina auxiliar se invoca de manera convencional, va a recibir los parámetros de la forma habitual.
  - Terminar proceso. Termina la ejecución de un proceso liberando sus recursos.
  - Escribir. Escribe un mensaje por la pantalla haciendo uso de la función “escribir_ker” proporcionada por el módulo HAL. Recibe como

parámetros la información que se desea escribir y su longitud. Devuelve siempre un 0.

La versión inicial de este módulo incluye una gestión de procesos básica que corresponde con un sistema monoprogramado. Para ser más precisos, hay que aclarar que en este sistema inicial, aunque se puedan crear y cargar en memoria múltiples programas, el proceso en ejecución continúa hasta que termina, ya sea voluntaria o involuntariamente debido a una excepción. Obsérvese que ninguna de las tres llamadas al sistema disponibles inicialmente puede causar que el proceso pase a un estado de bloqueado. A continuación, se presentan las principales características de la gestión de procesos en este sistema operativo inicial:

- La tabla de procesos ("tabla_procs") es un vector de tamaño fijo de BCPs.
- El BCP dispone de un puntero ("siguiente") que permite que el sistema operativo construya listas de BCPs que tengan alguna relación entre sí. En esta versión inicial sólo aparece una lista de este tipo, la cola de procesos listos ("lista_listos"), que agrupa a todos los procesos listos para ejecutar, incluido el que está ejecutándose actualmente.
- El tipo usado para la cola de listos (tipo "lista_BCPs") permite construir listas con enlace simple, almacenando referencias al primer y último elemento de la lista. Para facilitar su gestión, se ofrecen funciones que permiten eliminar e insertar BCPs en una lista de este tipo. Este tipo puede usarse para otras listas del sistema operativo (por ejemplo, para un mutex). Hay que resaltar que estas funciones están programadas de manera que, cuando se quiere cambiar un BCP de una lista a otra, hay que usar primero la función que elimina el BCP de la lista original y, a continuación, llamar a la rutina que lo inserta en la lista destino. Asimismo, conviene hacer notar que, por simplicidad, el uso de listas basadas en el tipo "lista_BCPs" exige que un BCP no pueda estar en dos listas simultáneamente. Si se quiere plantear un esquema en el que se requiera que un BCP esté en más de una lista, se deberá implementar un esquema de listas alternativo.
- La variable "p_proc_actual" apunta al BCP del proceso en ejecución. Como se comentó previamente, este BCP está incluido en la cola de listos.
- Con respecto a la creación de procesos, la rutina realiza los pasos típicos implicados en la creación de un proceso: buscar una entrada libre, crear el mapa de memoria a partir del ejecutable, reservar la pila del proceso, crear el contexto inicial, rellenar el BCP adecuadamente, poner el proceso como listo para ejecutar e insertarlo al final de la cola de listos.
- La liberación de un proceso cuando ha terminado voluntaria o involuntariamente implica liberar sus recursos (imagen de memoria, pila y BCP), invocar al planificador para que elija otro proceso y hacer un cambio de contexto a ese nuevo proceso. Nótese que, dado que no se va a volver a ejecutar este proceso, se especifica un valor nulo en el primer argumento de "cambio_contexto".
- Por lo que se refiere a la planificación, dado que la versión inicial de este módulo se corresponde con un sistema monoprogramado, el planificador (función "planificador") no se invoca hasta que termina el proceso actual. El algoritmo que sigue el planificador es de tipo FIFO: simplemente selecciona el proceso que esté primero en la cola de listos. Nótese que, si

todos los procesos existentes estuviesen bloqueados (situación imposible en la versión inicial), se invocaría la rutina “espera_int” de la que no se volvería hasta que se produjese una interrupción.

- Hay que resaltar que en este sistema operativo no existe un proceso nulo. Si todos los procesos existentes están bloqueados en un momento dado, lo que no es posible en la versión inicial, es el último en bloquearse el que se queda ejecutando la rutina “espera_int”. Esto puede resultar sorprendente al principio, ya que se da una situación en la que la cola de listos está vacía, pero sigue ejecutando el proceso apuntado por “p_proc_actual”, aunque esté bloqueado. La situación es todavía más chocante cuando termina el proceso actual estando los restantes procesos bloqueados, puesto que en este caso es el proceso que ha terminado el que se queda ejecutando el bucle de la función “planificador” hasta que se desbloquee algún proceso. Obsérvese que alguien tiene que mantener “vivo” al sistema operativo mientras no hay trabajo que hacer.

Los programas de usuario

En el subdirectorio “usuario” existen inicialmente un conjunto de programas de ejemplo que usan los servicios del minikernel. De especial importancia es el programa “init”, puesto que es el primer programa que arranca el sistema operativo. En un sistema real este programa consulta archivos de configuración para arrancar otros programas que, por ejemplo, se encarguen de atender a los usuarios (procesos de “login”). De manera relativamente similar, en nuestro sistema, este proceso hará el papel de lanzador de otros procesos, aunque en nuestro caso no se trata de procesos que atiendan al usuario, puesto que el sistema no proporciona inicialmente servicios para leer del terminal. Se tratará simplemente de programas que realizan una determinada labor y terminan. Como ocurre en un sistema real, los programas tienen acceso a las llamadas al sistema como rutinas de biblioteca. Para ello, existe una biblioteca estática, denominada “libserv.a”, que contiene las funciones de interfaz para las llamadas al sistema.

```
int crear_proceso(char *programa);
int terminar_proceso();
int escribir(char *texto, unsigned int longi)
```

Los programas de usuario no deben usar llamadas al sistema operativo nativo, aunque sí podrán usar funciones de la biblioteca estándar de C, como, por ejemplo, “strcpy” o “memcpy”.

La biblioteca “libserv.a” está almacenada en el subdirectorio “usuario/lib” y está compuesta de dos módulos:

- “serv”. Contiene las rutinas de interfaz para las llamadas. Se apoya en una función del módulo “misc” denominada “llamsis”, que es la que realmente ejecuta la instrucción de llamada al sistema. Para hacer accesible a los

programas una nueva llamada, el alumno deberá modificar este archivo para incluir la rutina de interfaz correspondiente.

- "misc". Como intenta indicar su nombre, este módulo contiene un conjunto diverso de funciones de utilidad. Entre ellas, la definición de la función "printf" que, evidentemente, se apoya en la llamada al sistema "escribir", de la misma manera que el "printf" en un sistema UNIX se apoya en la llamada "write". Asimismo, proporciona la función de conveniencia "llamsis". Esta función facilita la invocación de una llamada al sistema ocupándose de la tediosa labor de rellenar los registros con los valores adecuados y provocando, a continuación, el *trap*. A continuación, se muestra la estructura simplificada de esta función para que se pueda comprender mejor su funcionamiento:

```
int llamsis(int llamada, int nargs, ... /* argumentos */) {
    int i;
    escribir_registro(0, llamada);
    for (i=1; nargs; nargs--, i++)
        escribir_registro(i, args[i]);

    trap();
    return leer_registro(0);
}
```

Todos los programas de usuario utilizan el archivo de cabecera "usuario/include/servicios.h" que contiene los prototipos de las funciones de interfaz a las llamadas al sistema, así como el de la función "printf".

Pasos para la inclusión de una nueva llamada al sistema

Dado que parte de la labor de la práctica es incluir nuevas llamadas al sistema, se ha considerado oportuno incluir en esta sección los pasos típicos que hay que llevar a cabo en este sistema para hacerlo. Suponiendo que el nuevo servicio se denomina "nueva", estos son los pasos a realizar:

- Incluir en "minikernel/kernel.c" una rutina (que podría denominarse "sis_nueva") con el código de la nueva llamada.
- Incluir en "tabla_servicios" (fichero "minikernel/include/kernel.h") la nueva llamada en la última posición de la tabla.
- Modificar el fichero "minikernel/include/llamsis.h" para incrementar el número de llamadas disponibles y asignar el código más alto a la nueva llamada.
- Una vez realizados los pasos anteriores, el sistema operativo ya incluiría el nuevo servicio, pero sólo sería accesible desde los programas usando código ensamblador. Por lo tanto, es necesario modificar la biblioteca de servicios (fichero "usuario/lib/serv.c") para que proporcione la interfaz para el nuevo servicio. Se debería también modificar el fichero de cabecera que incluyen los programas de usuario ("usuario/include/servicios.h") para que dispongan del prototipo de la función de interfaz.

Problemas de sincronización dentro del sistema operativo

En la versión inicial, el código de las tres llamadas ejecuta todo el tiempo con las interrupciones habilitadas. Al ir añadiendo la funcionalidad pedida puede ser necesario revisar el código para hacer que ciertas zonas de código ejecuten con las interrupciones de un determinado nivel inhibidas.

Gracias al uso del mecanismo de interrupción software y los cambios de contexto involuntarios diferidos, característicos de los núcleos no expulsivos, no va a haber problemas de sincronización entre llamadas concurrentes. Sólo se presentarán problemas durante la ejecución de interrupciones. Concretamente, habrá que analizar los siguientes conflictos:

- El código de cada llamada al sistema (y de la rutina de tratamiento de la interrupción software) con la rutina del terminal y del reloj.
- El código de la rutina de tratamiento de la interrupción del terminal con la rutina del reloj.
- No habría que analizar posibles conflictos durante la ejecución de la rutina de tratamiento de la interrupción del reloj, ya que tiene máxima prioridad.