

# Sistemas Operativos

## 2º Curso I. T. Informática (Sistemas y Gestión) e I. Informática Práctica 4 – Diseño e implementación de sistemas operativos Convocatoria de Junio – Curso 2009/2010

En esta práctica el alumno debe modificar el sistema operativo OSO para añadirle nuevas llamadas al sistema. Los fuentes y la documentación del sistema operativo se encuentran en la página web de la asignatura, <http://ditec.um.es/so>. En esta práctica, lo importante no es sólo la correcta realización de los apartados, sino, sobre todo, alcanzar también una buena comprensión de todo el código fuente del sistema operativo OSO. Así, en la correspondiente entrevista el profesor podrá realizar las pertinentes preguntas a los alumnos sobre los distintos aspectos de la implementación del sistema operativo (y no sólo sobre la nueva funcionalidad añadida).

### Trabajo a realizar

Las llamadas al sistema que se deben implementar obligatoriamente se muestran a continuación. Para cada una de ellas se indica: el prototipo de C de la llamada al sistema, una breve descripción de lo que hace, el valor del registro AH (que indica al sistema operativo qué servicio se le está pidiendo), los parámetros de entrada y/o salida (junto con el registro o registros utilizados para cada parámetro) y el valor devuelto por la función (con el registro o registros en los que se devuelve el valor):

1. `int getpid(void)`

- Descripción: devuelve el PID del proceso invocador.
- Servicio: AH=03h.
- Entrada: ninguna.
- Salida: la función devuelve en AX el PID del proceso invocador.

2. `int initprocinfo(void)`

- Descripción: indica al núcleo que la siguiente llamada al sistema `getprocinfo` (ver a continuación) debe iniciar un nuevo recorrido de la tabla de procesos.
- Servicio: AH=04h.
- Entrada: ninguna.
- Salida: esta función siempre devuelve 0 en AX.

3. `int getprocinfo(struct procinfo far *info)`

- Descripción: permite obtener información de los procesos existentes. Cada vez que se invoca a esta función el núcleo devuelve información de un proceso diferente (internamente, el núcleo recorre la tabla de procesos de forma secuencial).
- Servicio: AH=05h.
- Entrada: la dirección de memoria de la estructura que recogerá la información (ES:BX).
- Salida: en caso de éxito, la función escribe en la dirección de memoria apuntada por ES:BX información asociada a un proceso y devuelve 0 en AX. Si se produce error (el núcleo ha llegado ya al final de la tabla de procesos), la función devuelve -1 en AX.
- Nota: la definición de `struct procinfo` debe ser:

```
struct procinfo {
    unsigned int    pid;           /* PID */
    char            nombre[13];   /* Nombre, devuelto como NOMBRE.EXT */
}
```

```

    unsigned int    segmento;    /* Segmento de memoria ocupado */
    unsigned int    estado;      /* Estado: 0 (listo), 1 (ejecución),
                                * 2 (bloqueado) y 3 (zombi) */
    unsigned long   tiempo;      /* Tiempo (en ticks) de ejecución total*/
}

```

**Nota.-** El estado zombi sólo tendrá que ser implementado por aquellos alumnos que, voluntariamente, implementen la llamada `waitpid` (apartado 11).

4. `int open(char far * nombre, unsigned char modo)`

- Descripción: abre un fichero que ya existe.
- Servicio: AH=06h.
- Entrada: nombre del fichero a abrir (ES:BX) y modo (AL) de apertura. El modo puede ser: 0 (sólo lectura), 1 (sólo escritura) y 2 (lectura/escritura)
- Salida: devuelve en AX el descriptor del fichero o -1 si se produjo error.
- Puntero de lectura/escritura del fichero: posición 0.

**Nota.-** Por completitud, se incluyen también los modos de escritura y lectura/escritura, aunque en el desarrollo de la práctica sólo se usará el modo de lectura.

5. `int read(int fd, void far * buffer, unsigned int contador)`

- Descripción: lee de un fichero.
- Servicio: AH=01h.
- Entrada: descriptor del fichero a leer (DX), buffer (ES:BX) y contador (CX).
- Salida: en AX, el número de bytes leídos ó -1 si se produjo error. Si ya se llegó al final del fichero devuelve 0.
- Puntero de lectura/escritura del fichero: avanza tantos bytes como se hayan podido leer con éxito.
- Nota: esta función ya está implementada para leer del teclado (`stdin`, cuando el descriptor es 0), pero hay que ampliarla adecuadamente para leer de ficheros (descriptores 3 en adelante).

6. `int close(int fd)`

- Descripción: cierra un fichero.
- Servicio: AH=07h.
- Entrada: descriptor del fichero a cerrar (DX).
- Salida: devuelve en AX 0 si se cerró con éxito, -1 en caso de error.

7. `long seek(int fd, long pos, unsigned char desde)`

- Descripción: coloca el puntero de lectura/escritura de un fichero.
- Servicio: AH=08h.
- Entrada: descriptor del fichero (DX), desplazamiento (dado por BX\*65536+CX) y punto relativo (AL). El punto relativo es: 0 (desde el comienzo del fichero), 1 (desde el final del fichero), 2 (desde la posición actual).
- Salida: la nueva posición (desde el comienzo del fichero) se devuelve en AX y DX como DX\*65536+AX. Si se produjo error el valor devuelto debe ser -1.
- Puntero de lectura/escritura del fichero: si no se produjo error se sitúa en la nueva posición indicada. El nuevo valor del puntero debe ser mayor o igual que 0 y menor o igual que el tamaño del fichero. Cualquier intento de establecer otro valor con `seek` debe producir error.

8. `void exit(int estado)`

- Descripción: termina al proceso invocador.
- Servicio: AH=09h.

- Entrada: el estado de terminación del proceso en CX.
- Salida: ninguna, esta función nunca regresa.

**Nota.-** El valor de terminación del proceso, en realidad, sólo será usado si se implementa el apartado voluntario 11 (llamada `waitpid`).

Adicionalmente, y de forma voluntaria, podrán implementarse también cualesquiera de las llamadas al sistema que se indican a continuación:

9. `int fork(void)`

- Descripción: crea un proceso hijo idéntico al padre pero *independiente* de él.
- Servicio: AH=0Ah.
- Entrada: ninguna.
- Salida: la función devuelve en AX el PID del proceso hijo en el caso del padre, 0 en el caso del hijo y  $-1$  si se produjo error.

10. `int exec(char far * nombre)`

- Descripción: el proceso invocador ejecuta un nuevo programa.
- Servicio: AH=0Bh.
- Entrada: nombre del programa a ejecutar (ES:BX).
- Salida: esta función no regresa en caso de éxito. Si se produce un error, devuelve  $-1$  en AX.

11. `int waitpid(int PID, int * estado)`

- Descripción: el proceso se bloquea a la espera de que termine el proceso hijo indicado.
- Servicio: AH=0Ch.
- Entrada: el PID del proceso a esperar (CX).
- Salida: estado de terminación del proceso (DX). La función devuelve en AX 0 si no hay error y  $-1$  si se produce error.

## Observaciones

Para comprobar la corrección de los distintos apartados, se proporcionará en el web una plantilla de corrección, con la que los alumnos deberán probar la corrección de su implementación (independientemente de cualesquiera otras pruebas adicionales que deseen realizar).

Se debe entregar una **memoria impresa con los listados y una breve explicación de los mismos** resaltando, por un lado, los puntos más importantes, y por otro, las mejoras introducidas. **Los ficheros se deben entregar en un disquete que acompañe a la memoria impresa. Dicho disquete deberá también arrancar OSO, y contener los ejecutables de los programas de prueba de las nuevas llamadas al sistema implementadas.**

La fecha tope de entrega para todas las ingenierías es el **lunes, 14 de junio.**

Murcia, 23 de abril de 2010.