Práctica B

Llamadas al Sistema

- **Objetivo**: entender las llamadas al sistema operativo que realizan los procesos de usuario
- **Llamadas al sistema**: mecanismo a través del cuál los procesos de usuario solicitan servicio al sistema operativo.
    - Supone cambiar la cpu de modo usuario a modo supervisor
    - Se realizan a través de interrupciones software
- **Interrupción software**: instrucción de código máquina del procesador que provoca las mismas operaciones que se hacen cuando se produce una interrupción hardware (externa o interna):
    1. Se ejecuta la instrucción *INT xx*
    2. El procesador cambia de modo usuario a modo privilegiado.
    3. Se accede a la tabla de vectores de interrupción (*TVI*). Se bifurca a la posición **(TVI+xx)*
    4. Se salvan los registros en una zona reservada del PCB. Entre ellos el PC y el SP.
    5. Se trata la interrupción.
    6. Se recuperan los registros y se retorna de la interrupción.
- **Llamadas al sistema en Minix:**
    - Al enlazar un programa de usuario se utilizan los ficheros de biblioteca de usuario cuyo código fuente se encuentra en */usr/src/lib*. Supongamos que nuestro programa de usario utiliza *fork()*
    - posix/_fork.c:fork() → _syscall(MM, FORK, &m)
        - Es decir, cuando invocamos desde un programa de usuario a la llamada al sistema *fork(),* se ejecuta la llamada a la función *_syscall()*. Sigamos la pista a esta función.
    - other/syscall.c:_syscall() → _sendrec()
        - Por lo tanto, la función *_syscall()* a su vez invoca a *_sendrec()*
        - Conviene indicar en este punto que, para la misma función, también se utiliza syslib/taskcall.c:_taskcall()
    - i386/rts/_sendrec.s:_sendrec() → **int SYSVEC**
        - Vemos ahora que la implementación de la función *_sendrec()* se realiza en ensamblador. Observe de qué forma tan natural se puede unir la programación en C y en ensamblador. Note cómo ahora el directorio se llama i386, es decir, es un nombre asociado a una arquitectura concreta de procesador. En este punto el código es específico del procesador.

        - La función *__sendrec()* comienza a partir de la línea 43. Vemos que tiene muy pocas instrucciones (push=insertar en pila, mov=movimiento entre memoria y registros, pop=extraer de la pila). La instrucción más importante es **int**.
        - Esta es la **interrupción software: int SYSVEC**
        - Por lo tanto lo que hace esta función es poner en ciertos registros los argumentos que trae de las funciones anteriores y, a continuación, realizar la llamada al sistema. Como vemos el argumento de esta llamada al sistema es 33.
        - A partir de este punto el procesador cambia a **modo privilegiado**
    - src/kernel/protect.c:prot_init(): en esta función se inicializan la tabla de vectores de interrupción. Entre ellos SYS386_VECTOR.
        - En el fichero src/kernel/const.h se define la constante SYS386_VECTOR con el valor de SYSVEC
        - La recepción de esta interrupción software se asocia a la función *s_call()***.** Vea la línea 206. Sigamos por lo tanto la pista a la función *s_call()*
    - src/kernel/mpx386.s:*s_call()* → *sys_call()*
        - Seguimos en ensamblador. La función *s_call()* invoca a *sys_call()* que ya está escrita en C. Observe cómo los argumentos para *sys_call()* se han insertado previamente en la pila (m_ptr, src/dest, SEND/RECEIVE/BOTH).
    - src/kernel/proc.c:sys_call(). Implementación de la llamada al sistema.
        - Todas las funciones que implementa Minix son requeridas y respondidas mediante mensajes.
        - En sys_call() se envía el mensaje al servidor correspondiente. En nuestro ejemplo el servidor es MM y el tipo de mensaje es FORK.
    - src/kernel/mpx386s.s:s_call() → _restart()
        - Cuando sys_call() termina, el mensaje ya ha sido enviado, en este caso a MM
        - Se ejecuta _restart() que permitirá continuar con la ejecución de otro proceso, normalmente éste será MM ya que en este momento tiene un mensaje pendiente de tratamiento. Es en este punto donde el procesador cambia de modo privilegiado a **modo usuario**
- **Recepción del mensaje en el servidor MM**: ver /usr/src/mm
    - En main.c:main() está el código del gestor de memoria (MM). Una vez realizada la inicialización (mm_init()) se entra en un bucle infinito en el que el servidor se queda esperando la recepción de un mensaje (get_work()).
    - Cuando se recibe el mensaje se invoca a la función correspondiente almacenada en un vector de funciones llamado *call_vec[]*.
- **Hacer**:
    1. Repase todo el enunciado de esta práctica. Localice todas y cada una de las líneas de código

que se mencionan. Debe tener muy claro qué código se ejecuta en modo usuario y qué código se ejecuta en modo privilegiado.

2. Localice la función del gestor de memoria que se ejecuta cuando se recibe la llamada al sistema fork(). Justifique cómo ha localizado dicha función. Modifique esa función para que salude. Compruebe que saluda. Explique los resultados.