

Objetivos

- Examinar la implementación de **drivers en Xinu**.
- Implementar funciones de alto nivel para un **controlador de periférico**.

Referencias

- [1] Tanenbaum, Bos – Modern Operating Systems - Prentice Hall; 4 edition (March 10, 2014) - ISBN-10: 013359162X
- [2] Douglas Comer - Operating System Design - The Xinu Approach. CRC Press, 2015. ISBN : 9781498712439
- [3] Silberschatz, Galvin, Gagne - Operating Systems Concepts - John Wiley & Sons; 10 edition (2018) – ISBN 978-1-119-32091-3

Software y Hardware

Linux y Xinu. La versión de Xinu que utilizamos es para arquitectura PC (x86). Ejecutamos el sistema operativo Xinu en una máquina virtual llamada QEMU, que emula una PC básica.

El trabajo puede realizarse sobre las máquinas de los laboratorios (RECOMENDADO).

Quienes tengan Linux en sus casas, podrían intentar instalar todo lo necesario y llevarlo a cabo ahí también. Una tercera posibilidad es el acceso remoto RDP comentado en la web de la materia.

Ejercicio 1. Subsistema de E/S (I/O). Desarrollo del driver del teclado ps/2 de una PC.

El sistema operativo XINU (xinu-pc) cuenta con un incompleto driver para el teclado. En particular, el lower-half del driver está implementado, pero no está integrado con la sección upper-half. También falta la implementación de la sección upper-half.

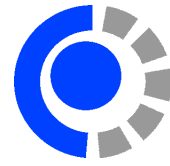
La función init() ya existe, y el sistema operativo Xinu ejecuta la misma cuando inicia, configurando los registros del controlador del teclado y la rutina de atención de interrupciones.

La rutina de atención de interrupciones actual simplemente “muestra en pantalla” un código hexadecimal para cada tecla pulsada, o liberada. El teclado genera, por cada tecla, dos códigos: uno para cuando la tecla es pulsada, y otro cuando la tecla es liberada.

- a. Como empleado del mes de Shigeru Miyamoto, se le solicita que complete correctamente el software del driver del teclado.

Las características que debe tener el driver son:

- **open(), read(), close() paradigma.** Implementar getc() también. read() debe utilizar getc().
- **read() :** si bien la interfaz de read() permite solicitar varios bytes a ser colocados en un buffer del usuario, implementar únicamente la solicitud de un byte, para simplificar. Un proceso no puede realizar un read() o getc() si no realizó un open() exitoso. getc() devuelve el código que identifica a la tecla pulsada o liberada;
- **Buffering interno del driver :** Un buffer de 10 entradas. Si el buffer se llena y arriban nuevas entradas desde el teclado, el driver debe descartar las nuevas entradas.
Implementación : Una implementación del buffer posible es con un arreglo, implementado FIFO,



un semáforo para conocer si hay espacio libre o no en el buffer, e índices para conocer el inicio del buffer y el final del buffer.

- Sólo un proceso puede obtener acceso al teclado. El control es a través de `open()` y `close()`. Los demás procesos requiriendo acceso deberán esperar.

Tareas que son requeridas y visibles para este laboratorio. Será necesario remover el código que ya no se utilice. Por ejemplo, la ISR ya no debe mostrar en pantalla el código hexadecimal (aunque puede utilizarlo para conocer qué códigos hexadecimales utiliza cada tecla, y mapear esos códigos a nombres más sencillos en el código fuente; si se necesita).

La sincronización entre el **upper-half** del driver del teclado (`kbdgetc`) y el **lower-half** (la rutina de atención de interrupciones del driver del teclado `kbhandler`) es que `kbdgetc()` bloquea esperando al semáforo del buffer FIFO (bloquea si en el buffer fifo no hay evento que consumir). Y estará bloqueado hasta que `kbhandler()` se active por una interrupción, y agregué un evento al buffer FIFO, y active un semáforo de sincronización del buffer FIFO. Cuando eso suceda desbloqueará a `kbdgetc()` si se encontraba “bloqueado/waiting”.

- b. Modifique el juego de Galaga para que utilice el nuevo driver del teclado.
- Agregue al gálaga un proceso más. Este proceso simplemente realiza un `open()` del teclado, y luego se quedará por siempre realizando `read()` del teclado, solicitando una pulsación de tecla. Cabe aclarar, que este proceso, al realizar el `read()`, y si no hay teclas en el buffer del driver del teclado, será bloqueado por el kernel XINU, hasta que se “presione” o se “suelte” una tecla.
- Luego del `read()`, este proceso podría “actualizar” la variable `tecla_actual`, que el videojuego Gálaga ya estaba utilizando para verificar eventos de teclas, y de esta manera, no será necesario realizar más modificaciones.

El proceso realizará un `read` similar al código siguiente:

```
char c;  
open(KEYBOARD, etc);  
while (1) {  
    read(KEYBOARD, &c, 1);  
    Etc;  
}
```