Objetivos

- Analizar diferentes mecanismos de **sincronización de procesos.**
- Analizar diferentes mecanismos de **comunicación entre procesos**.

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. *Sincronización entre procesos.*

a. Agregar este programa al shell de Xinu (recuerde cambiar el nombre de main por el nombre que seleccione para el programa):

```
#include <xinu.h>

void    produce(void), consume(void);

int32   n = 0;          /* Global variables are shared by all processes   */

/*------------------------------------------------------------------------
 *  main  -  Example of unsynchronized producer and consumer processes
 *------------------------------------------------------------------------
 */
void    main(void)
{
        resume( create(consume, 1024, 20, "cons", 0) );
        resume( create(produce, 1024, 20, "prod", 0) );
}

/*------------------------------------------------------------------------
 * produce  -  Increment n 2000 times and exit
 *------------------------------------------------------------------------
 */
void    produce(void)
{
        int32   i;

        for( i=1 ; i<=2000 ; i++ )
                n++;
}

/*------------------------------------------------------------------------
 * consume  -  Print n 2000 times and exit
 *------------------------------------------------------------------------
 */
void    consume(void)
{
        int32   i;

        for( i=1 ; i<=2000 ; i++ )
                printf("The value of n is %d \n", n);
}
```

b. Este programa es un típico programa compuesto de 3 procesos, main, productor, consumidor. Los procesos productor y consumidor se “comunican” a través de una variable compartida. El productor genera valores, y el consumidor los muestra en pantalla. Ejecutar el programa. Describir qué sucede. ¿El programa funciona como el programador pretendió?
c. Modifique el programa utilizando algún mecanismo provisto por el sistema operativo para la sincronización de procesos, de manera que el programa funcione como el programador pretendía.

Ejercicio 2. *Implementación de recursos lógicos.*

a. Agregar este programa al shell de Xinu.

```
/* mut.c - mut, operar, incrementar */
#include <xinu.h>
```

```
void    operar(void), incrementar(void);

unsigned char x = 0;

/*------------------------------------------------------------------------
 * mut  --  programa con regiones criticas
 *------------------------------------------------------------------------
 */
void    mut(void)
{
        int i;

        resume( create(operar, 1024, 20, "process 1", 0) );
        resume( create(incrementar, 1024, 20, "process 2", 0) );

        sleep(10);

}


/*------------------------------------------------------------------------
 * operar x e y
 *------------------------------------------------------------------------
 */
void    operar(void)
{
        int y = 0;

        printf("Si no existen mensajes de ERROR entonces todo va OK! \n");

        while (1) {

                /* si x es multiplo de 10 */
                if ((x % 10) == 0) {

                        y = x * 2;       /* como y es el doble de x entonces
                                          * y es multiplo de 10 tambien
                                          */

                        /* si y no es multiplo de 10 entonces hubo un error */
                        if ((y % 10) != 0)
                                printf("\r ERROR!! y=%d, x=%d \r", y, x);
                }
        }
}

/*------------------------------------------------------------------------
 * incrementar x
 *------------------------------------------------------------------------
 */
void    incrementar(void)
{
        while (1) {
                x = x + 1;
        }
}
```

Ejecutar el programa. ¿Por qué este programa (sus procesos) emiten mensajes de error? ¿Qué es posible hacer para solucionar el problema?.

b. El sistema operativo Xinu no provee mecanismos para resguardar regiones críticas. Pero, su autor, define a Xinu como un microkernel, lo que significa que provee mecanismos mínimos que permiten luego construir otros más complejos. Usando el mecanismo de semáforos provisto por Xinu, implementar una protección de exclusión mutua (mutex). Este mutex debe estar compuesto por dos funciones :
mutex_init(), mutex_lock(); y mutex_unlock();

c. Proteger las regiones críticas del programa original con este nuevo mecanismo de exclusión mutua.

Ejercicio 3. *Implementar un programa en Linux que utilice* ***memoria compartida (comunicación entre procesos).***

a. Desarrollar un programa A que solicite al sistema operativo Linux una región de memoria compartida utilizando POSIX shared memory. Luego, el proceso A debe abrir el archivo /usr/share/doc/aufs-dkms/filesystems/aufs/design/06mmap.txt
(archivo de texto) , leer su contenido, y colocarlo en la región de memoria compartida creada.

COMPILAR CONTRA la BIBLIOTECA librt así:

gcc -o proceso_a proceso_a.c -lrt

b. Desarrollar un segundo programa, B, que le solicite a Linux el acceso a la memoria compartida, y presente en pantalla el contenido de esa región (la cual será el contenido del archivo de texto).
COMPILAR COMO EN a.

c. Utilice las funciones open(), read(), close() en a. para leer el contenido del archivo.

Documentación:
man 2 open
man 2 read
man 2 close
man 3 shm_overview
man 7 shm_open
man 3 shm_unlink

Ejercicio 4. *Implementar un programa en Xinu que utilice* ***pasaje de mensajes (comunicación entre procesos).***

Usted ha sido contratado por Shigeru Miyamoto para implementar en Xinu un video juego.
Él le explica que ha portado una versión de Galaga de la consola Game Boy Advance (GBA) a Xinu, y le solicita que lo termine.
Luego de una revisión usted detecta que el juego está lleno de bugs y hacks.
La paleta de colores es incorrecta, los disparos no siempre alcanzan al objetivo correctamente (detector de colisiones defectuoso), no existe puntuación, el código es horrible.

NOTA: el juego está en https://se.fi.uncoma.edu.ar/so/misc/xinu-pc-galaga.tar.gz
(incluido en el shell de Xinu. Ejecute galaga en el shell, y vuelva con ctrl+alt+1 a la interfaz gráfica).
Las teclas predeterminadas en el juego son: z (inicia el juego), a (izquierda), s (derecha), j (dispara).

a. Divida el videojuego en al menos 3 procesos:
   - Un proceso 1 es el actual juego galaga.
   - Un proceso 2 mantiene las vidas y el puntaje.
   - Un proceso 3 control.

b. El proceso 3 control crea los otros dos procesos, y se queda esperando por un mensaje de finalización.
c. El proceso 2 debe mostrar en la pantalla (sobre el tapiz amarillo de fondo de la interfaz grafica de Xinu las vidas que le queda al jugador, y el puntaje. Este proceso se queda esperando mensajes.
d. El proceso 1 es el juego actual. Este proceso utilizará comunicación inter-procesos de Xinu, enviando mensajes:
    i. Cuando el proceso 1 detecta que la nave player es alcanzada por una nave enemiga (colisión) el proceso 1 le envía un mensaje al proceso 2 para indicar que el player perdió una vida.
    ii. Cuando el proceso 1 detecta que un disparo colisionó con una nave enemiga entonces el proceso 1 le envía un mensaje al proceso 2 para que aumente el puntaje de player.
    iii. Cuando el proceso 1 detecta que el jugador quiere terminar (por ej. el jugador presiona cierta tecla -elija UD. la tecla: observe como es el sistema de pulsaciones el teclado en el juego-) entonces el proceso 1 le envía al proceso 3 un mensaje de que el juego terminó y que el jugador quiere salir del juego.

e. Si el proceso 3 control obtiene un mensaje de salir debe finalizar todo el juego (todos los procesos relacionados con el mismo).
f. Bonus (optativo): agregar si el jugador perdió, ganó, matar bugs. Cualquier otra característica que ponga feliz a Shigeru Miyamoto.