

# Trabajo Práctico 0

## Bibliografía:

Brian Kernighan y Dennis Ritchie, El lenguaje de programación C  
(título original en inglés: The C Programming Language)

Eduardo Grosclaude, Taller de Lenguaje en C  
<http://se.fi.uncoma.edu.ar/pse2020/taller-c/index.html>

0. Escriba un programa hello world. Compilar y ejecutar.

Por ejemplo:

```
vi hello.c # edicion. Usar vi u otro editro de predileccion
make hello # compilacion
gcc -o hello hello.c # compilación invocando gcc
./hello    # ejecucion
```

```
#include <stdio.h>
#include <stdlib.h>    /* para las funciones system y exit */

int main() {
    printf ("Hello Word\n");
}
```

**1) Enumerar los tipos de datos básicos en C. Escriba una definición de una variable de cada tipo. Por ej. :**  
**int a;**

Los tipos básicos son:

1. int: almacena valores enteros (32 bits)
  - Rango de representación: [-2,147,483,648 / 2,147,483,647]
2. float: almacena valores en punto flotante de precisión simple (32 bits)
  - Rango de representación:  $\pm 3.4 \times 10^{38}$  (precisión de 6 dígitos decimales)
3. double: almacena valores en punto flotante de precisión doble (64 bits)

- Rango de representación:  $\pm 1.7 \times 10^{308}$  (precisión de 15 dígitos decimales)

4. char: almacena caracteres individuales (8 bits)

- Rango de representación: [-128 / 127]

6. long: almacena enteros largos (32 bits)

- Rango de representación: [-2,147,483,648 / 2,147,483,647]

- char (un elemento del tamaño de un byte) puede tomar los valores [-128,127]
- int (un número entero con signo)
- long (un entero largo)
- float (un número en punto flotante)
- double (un número en punto flotante, doble precisión)

```
char letra = 'c' ;
float temp, division;
long int largo;
double resultado;
unsigned int num;
char frase[] = "hola mundo";
int arreglo[20];
```

Cuando declaramos una variable o forzamos una conversión de tipo, utilizamos una especificación de tipo. Ejemplos de declaración de variables:

```
char a;
int alfa,beta;
float x1,x2;
```

Los **tipos enteros** (char, int y long) admiten los modificadores signed (con signo) y unsigned (sin signo). Un objeto de datos **unsigned** utiliza todos sus bits para representar magnitud; unsigned utiliza un bit para signo, en representación complemento a dos.

El modificador signed sirve sobre todo para explicitar el signo de los chars. El default para un int es signed; el default para char puede ser signed o unsigned, dependiendo del compilador.

```
unsigned int edad;
signed char beta;
```

Un int puede afectarse con el modificador short (corto).

```
short i;
unsigned short k;
```

Cuando en una declaración aparece sólo el modificador unsigned o short, y no el tipo, se asume int. El tipo entero se supone el tipo básico manejable por el procesador, y es el tipo por omisión en varias otras situaciones. Por ejemplo, cuando no se especifica el tipo del valor devuelto por una función.

El modificador long puede aplicarse también a float y a double. Los tipos resultantes pueden tener más precisión que los no modificados. long float e; long double pi;

## 2) Explique qué significa la palabra word cuando el contexto es la Arquitectura de una Computadora.

El tamaño posible de los objetos datos y/o direcciones de memoria en la arquitectura(?)

El tamaño de una palabra, o word, puede variar dependiendo de la arquitectura de la computadora, donde este puede consistir en 32 bits, como en la arquitectura MIPS.

## 3) ¿Cuántos bits puede almacenar una variable declarada como int en C? (ej. int var;) Suponga 3 computadoras de arquitecturas diferentes: arquitecturas de 8 bits, 32 bits y 64 bits?. ¿Y en su PC?

NOTA: la respuesta no es tan trivial como parece. Deberá buscar información sobre el lenguaje de programación C. Use wikipedia u otros recursos.

En C, una variable de tipo int puede tomar [-32768,32767]

## 4) Explique la diferencia en la salida de las siguientes instrucciones:

```
char a = 'M';  
printf ("a = %i \n", a); // Imprime el valor en int  
printf ("La letra %c \n", a); // Imprime el carácter M
```

¿Cuál es el valor numérico de a?

Salida:  
a = 77  
La letra M

Ayuda: En la función "printf()" se pasa como parámetro la cadena de caracteres que se desea imprimir y, además, cada una de las variables que serán visualizadas.

Si deseamos imprimir más de una variable en una cadena de caracteres, el orden de los parámetros debe corresponder al orden de dichas variables en la cadena.

```
char sensor;  
float temp;  
printf('La temperatura de %c es %f',sensor, temp);
```

Salida: La temperatura de M es 3.140000

## 5. ¿Cuál es el rango numérico de i y j en su PC?

```
char i;  
unsigned char j;
```

El rango del char es 1 byte sería [-128, 127] y el rango del unsigned char es 1 serían [0, 255].

## 6. ¿Cuál es el valor en base 2 (binario) de i, j, k?

```
char i = 'a';
```

```
char j = 77;  
char k = 0x4D;
```

- Para i = 'a':  
El valor de 'a' en ASCII es 97.  
 $97_{10} = 1100001_2$
- Para j = 77:  
El valor de j ya está en decimal, así que solo necesitamos convertirlo a binario:  
 $77_{10} = 1001101_2$
- Para k = 0x4D  
0x4D es una notación hexadecimal, que representa el número 77 en decimal.  
Por lo tanto, k es igual a j, y su valor en binario también es 1001101.

## 7. ¿Cuáles de estas declaraciones contienen errores?

```
integer a;    // ERROR  
short i,j,k;  
long float (h);           // ERROR  
double long d3;    // ERROR  
unsigned float n;    // ERROR  
char 2j;    // ERROR  
int MY;  
float ancho, alto, long; // ERROR  
bool i;      // ERROR
```

## 8. Averigüe los tamaños de todos los tipos básicos en su sistema aplicando el operador sizeof().

```
#include <stdio.h>  
#include <stdlib.h>    /* para las funciones system y exit */  
int main() {  
    printf("El tamaño de char es: %zu bytes\n", sizeof(char));  
    printf("El tamaño de short es: %zu bytes\n", sizeof(short));  
    printf("El tamaño de int es: %zu bytes\n", sizeof(int));  
    printf("El tamaño de long es: %zu bytes\n", sizeof(long));  
    printf("El tamaño de long long es: %zu bytes\n", sizeof(long long));  
    printf("El tamaño de float es: %zu bytes\n", sizeof(float));  
    printf("El tamaño de double es: %zu bytes\n", sizeof(double));  
    printf("El tamaño de long double es: %zu bytes\n", sizeof(long double));  
    printf("El tamaño de _Bool es: %zu bytes\n", sizeof(_Bool));  
}
```

Salida por Pantalla:

El tamaño de char es: 1 bytes

El tamaño de short es: 2 bytes  
El tamaño de int es: 4 bytes  
El tamaño de long es: 8 bytes  
El tamaño de long long es: 8 bytes  
El tamaño de float es: 4 bytes  
El tamaño de double es: 8 bytes  
El tamaño de long double es: 16 bytes  
El tamaño de \_Bool es: 1 bytes

9. Prepare un programa con una variable de tipo char y otra de tipo unsigned char. Inicialice ambas variables con los valores máximos de cada tipo, para comprobar el resultado de incrementarlas en una unidad. Imprima los valores de cada variable antes y después del incremento.

```
#include <stdio.h>
#include <stdlib.h>    /* para las funciones system y exit */

int main() {
    unsigned char var1 = 255; // unsigned = para que no tenga signo
    char var2 = 127;
    printf("unsigned char = %i y char es %i \n",var1, var2);
    var1 = var1 + 1;
    var2 = var2 + 1;
    printf("unsigned char + 1 = %i y char + 1 es %i \n",var1, var2);
}
```

Salida:

```
unsigned char = 255 y char es 127
unsigned char + 1 = 0 y char + 1 es -128
```

Por lo tanto no da error sino que pega la vuelta.

10.a ¿Qué hace falta corregir para que la variable r contenga la división exacta de a y b?

```
int a, b;
float r;
a = 5;
b = 2;
r = a / b;
```

```
#include <stdio.h>
```

```
#include <stdlib.h>    /* para las funciones system y exit */
```

```
int main() {

    float a, b;
    float r;
    a = 5;
    b = 2;
    r = a / b;
```

```
    printf("r = %f \n",r );  
}
```

Se debe hacer flotante a r o castear.

Otra forma:

```
#include <stdio.h>  
#include <stdlib.h>    /* para las funciones system y exit */  
  
int main() {  
  
    int a,b;  
    float r;  
    a = 5;  
    b = 2;  
    r = (float)a/b;  
    printf("resultado:%f\n", r);  
}
```

10.b ¿Qué resultado puede esperarse del siguiente fragmento de código?

```
int a, b, c, d;  
a = 1;  
b = 2;  
c = a / b;  
d = a / c;
```

Da excepción de coma flotante ya que c y d tienen que ser flotante.

10.c ¿Cuál es el resultado del siguiente fragmento de código? Anticipe su respuesta en base a lo dicho en esta unidad y luego confirme mediante un programa.

```
printf("%d\n", 20/3);  
printf("%f\n", 20/3);  
printf("%f\n", 20/3.);  
printf("%d\n", 10%3);  
printf("%d\n", 3.1416);  
printf("%f\n", (double)20/3);  
printf("%f\n", (int)3.1416);  
printf("%d\n", (int)3.1416);
```

Salida:

```
6  
0.000000  
6.666667  
1  
1440612368  
6.666667
```

6.666667

3

11. Escribir un programa que multiplique e imprima  $100000 * 100000$ . ¿De qué tamaño son los ints en su sistema?

```
#include <stdio.h>
#include <stdlib.h>      /* para las funciones system y exit */

int main() {

    int a, b, c, d;
    a = 100000;
    b = 100000;
    c = a * b;
    printf("100000* 1000000 = %i",c);
}
```

Salida:

100000\* 1000000 = 1410065408

Por lo tanto llego al limite. El límite de los ints son 4 bytes

12. Descargue el código ahorcado.c propuesto por la cátedra.

```
#include <stdio.h>
#include <stdlib.h>      /* para las funciones system y exit */

int main() {

    int c;

    /* Decirle al sistema que el modo input es RAW */
    system ("/bin/stty raw");

    while(1) {
        printf("\r
        printf("\r c = %c  ingrese una letra (0 para salir): ", c);
        c = getchar();

        if (c == '0')
            break;
    }

    system ("/bin/stty sane erase ^H");
}
```

12.a Investigar cuál es la función que cumplen las siguientes líneas de código:

```
#include <stdio.h>
```

Sirve para indicar al compilador que incluya otro archivo. Cuando en compilador se encuentra con esta directiva la sustituye por el archivo indicado. En este caso es el archivo `stdio.h` que es donde está definida la función `printf`

```
#include <stdlib.h>
```

`stdlib.h` es el archivo de cabecera de la biblioteca estándar de propósito general del lenguaje de programación C. Contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras.

```
system("/bin/stty raw");
```

Esta línea ejecuta el comando del sistema `/bin/stty raw`, que configura el terminal en modo "raw". En el modo raw, los caracteres ingresados por el usuario no son procesados por el sistema operativo hasta que se presiona Enter. Esto significa que cada tecla presionada por el usuario es inmediatamente entregada al programa en ejecución, sin esperar por el buffer de entrada estándar

```
system("/bin/stty sane erase ^H");
```

Esta línea ejecuta el comando del sistema `/bin/stty sane erase ^H`, que restaura la configuración del terminal a un estado "sano" (`sane`). También configura el carácter de retroceso (`backspace`) a `^H`. Esto es importante porque después de usar el modo "raw", es posible que el terminal quede en un estado no deseado, por lo que esta línea se asegura de restaurar la configuración original del terminal después de que el programa haya terminado su ejecución. El carácter `^H` es el código ASCII para el carácter de retroceso (`backspace`). Esto indica que cuando el usuario presione la tecla de retroceso, se enviará `^H` al programa, lo que generalmente representa el carácter de retroceso.

12.b Complete el código `ahorcado.c` usando `printf()` y `getchar()`, para desarrollar el juego del ahorcado.

```
#include <stdio.h>
#include <stdlib.h> /* para las funciones system y exit */
#include <string.h> /* para la función strlen */

//juego del ahorcado.

int main() {
    const int MAX_INTENTOS = 6; //guarda la cantidad de intentos posibles. No cambia.
    const char PALABRA_SECRETA[] = "AHORCADO"; //Guarda la palabra a adivinar como un
    arreglo de caracteres.
    int intentos = 0; //Intentos actuales del juego
    char letra; //carácter actual puesto por el usuario.
    int palabra_len = strlen(PALABRA_SECRETA); //Cantidad de caracteres.
    char palabra_mostrada[palabra_len + 1]; //Palabra mostrada en pantalla. Arranca sin
    letras.
    int i; //contador.

    // Inicializar la palabra mostrada con guiones bajos
    for (i = 0; i < palabra_len; i++) {
        palabra_mostrada[i] = '_';
    }
    palabra_mostrada[palabra_len] = '\0'; // Agregar el carácter nulo al final de la
    cadena

    /* Decirle al sistema que el modo input es RAW */
    system("/bin/stty raw"); //

    printf("\n¡Bienvenido al juego del Ahorcado!\n");
    printf("Adivina la palabra secreta:\n");
```



```

while (1) {
    printf("\nPalabra: %s\n", palabra_mostrada);
    printf("Intentos restantes: %d\n", MAX_INTENTOS - intentos);
    printf("\r                                     ");
    printf("ingrese una letra (0 para salir): ");
    letra = getchar();

    if (letra == '0') // Salir del juego si se ingresa '0'
        break;

    int acierto = 0; //controla si fue acierto o no. 0 = No hay acierto . 1 =
Hay acierto.
    for (i = 0; i < palabra_len; i++) {
        if (PALABRA_SECRETA[i] == letra) { //si el caracter de la palabra a
adivinar es igual a la letra que se ingresó...
            palabra_mostrada[i] = letra; //Se guarda en la palabra mostrada.
            acierto = 1; //Hay acierto.
        }
    }

    if (!acierto) { //Si no hay acierto, se consume un intento.
        intentos++;
        printf("\r                                     ");
        printf("La letra '%c' no está en la palabra.\n", letra);
    }

    // Verificar si se han agotado los intentos
    if (intentos >= MAX_INTENTOS) {
        printf("\n¡Perdiste! La palabra secreta era: %s\n", PALABRA_SECRETA);
        break;
    }

    // Verificar si se ha adivinado la palabra
    int acertado = 1; //Controla si se acertó la palabra completa o no. 0 = No se
adivinó toda la palabra secreta. 1 = Se adivinó toda la palabra secreta.
    for (i = 0; i < palabra_len; i++) {
        if (palabra_mostrada[i] != PALABRA_SECRETA[i]) { //Si hay un carácter
distinto, entonces todavia no se acertó.
            acertado = 0;
            break;
        }
    }

    if (acertado) { //Si se adivinó la palabra secreta, termina el juego.
        printf("\n¡Felicidades! ¡Adivinaste la palabra secreta!\n");
        break;
    }

}

system("/bin/stty sane erase ^H"); //Restaura la configuración del terminal a un
estado "sano" (sane)

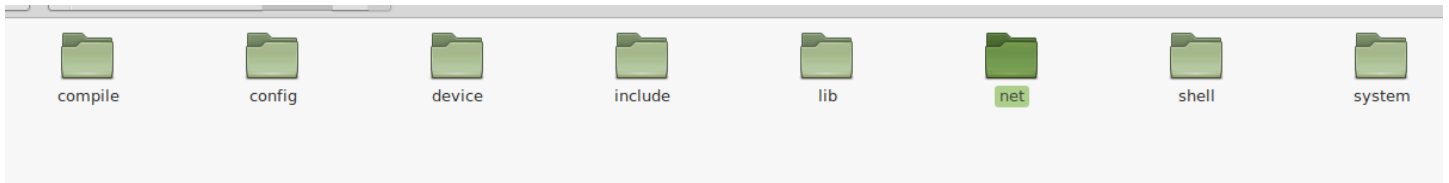
```

}

## Trabajo Práctico 1

- 1) Descargue el código fuente de Xinu para PC, desde la web de la materia. El sistema operativo ya contiene, también, el código fuente del shell de Xinu. Compilar y verificar el funcionamiento de Xinu utilizando las instrucciones en la web de la materia.
  - ¿Qué componentes trae esta versión de Xinu? ¿Qué periféricos de PC son accesibles desde QEMU? ¿Cómo se accede al puerto serie de la PC en QEMU, el cual permite utilizar el shell? ¿Cuántos procesos existen en ejecución? ¿Cómo lo obtuvo?

Los componentes que trae esta versión de XINU son



(revisar)

los periféricos de PC que son accesibles desde QEMU son el mouse, teclado, disco duro, interfaz de red, controlador de sonido, ..

(revisar)

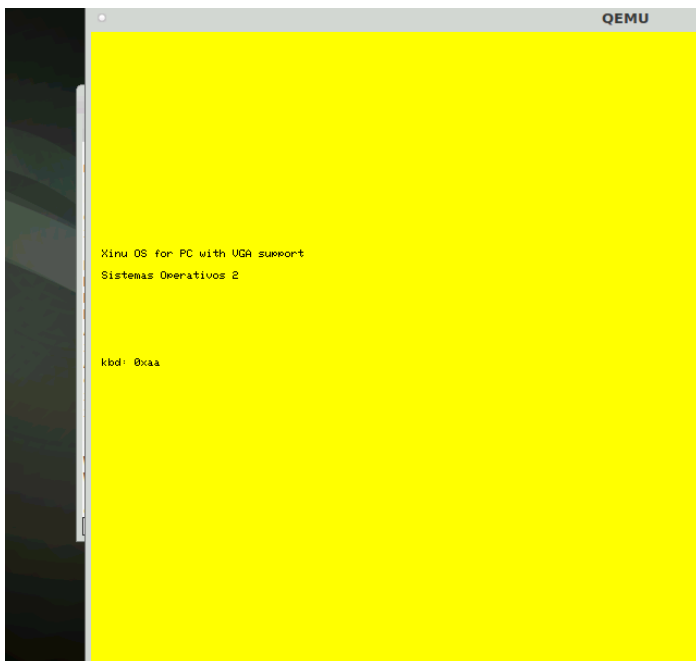
Para acceder al puerto serie de la PC en QEMU, el cual permite utilizar el shell, se utiliza el comando ctrl+alt+3.

Procesos en ejecución:

| Pid | Name         |
|-----|--------------|
| 0   | prnull       |
| 1   | rdsproc      |
| 3   | Main process |
| 4   | shell        |
| 6   | ps           |

Se obtiene a través del comando "ps" dentro del shell de XINU.

- 2) Modifique Xinu. Editar el archivo main.c y emita un mensaje a la pantalla gráfica a colores VGA de PC. Compilar y verificar.



3) Incorporación de un programa al shell de Xinu. Agregue un nuevo programa a Xinu. Deberá incorporarlo al shell de Xinu de la siguiente manera:

1. Escriba un programa hello world en un archivo .c debajo del directorio shell/. (Atención: en Xinu la función principal de un programa no debe llamarse main() ). Ejemplo de código de programa:

```
#include <xinu.h>
mi_programa()
{
printf("Hola mundo! \n");
}
```

```
#include <xinu.h>
#include <stdio.h>
xsh_programita() {
printf("buenas \n");
}
```

2. Incorpore su programa a la estructura que define los programas del shell, dentro del archivo shell/cmdtab.c y en include/shprototypes.h

cmdtab.c

```
{ "udpecho", FALSE, xsh_udpecho},
{ "udpeserver", FALSE, xsh_udpeserver},
{ "uptime", FALSE, xsh_uptime},
{ "?", FALSE, xsh_help},
{ "programita", FALSE, xsh_programita}
```

shprototypes.h

```
extern shellcmd xsh_help (int32, char
/* in file xsh_programita.c */
extern shellcmd xsh_programita ();
```

3. Compilar y verificar que su programa se encuentra incorporado al sistema y puede ejecutarlo.

```
xsh $ programita
buenas
```

4. ¿Por qué en Xinu no se llama main() la función principal de cada nuevo programa?

Porque "main.c" ya esta creada y esta configurada para usarla dentro de la salida VGA de XINU.

Ejercicio 4. Creación de procesos en Xinu. Incorpore a su programa anterior código para crear los siguientes dos procesos que se observan en el ejemplo:

```
/* ex2.c - main, sndA, sndB */
#include <xinu.h>

void  sndA(void), sndB(void);

/*-----
 * main  --  example of creating processes in Xinu
 *-----
 */
void  main(void)
{
    resume( create(sndA, 1024, 20, "process 1", 0) );
    resume( create(sndB, 1024, 20, "process 2", 0) );
}

/*-----
 * sndA  --  repeatedly emit 'A' on the console without terminating
 *-----
 */
void  sndA(void)
{
    while( 1 )
        putc(CONSOLE, 'A');
}

/*-----
 * sndB  --  repeatedly emit 'B' on the console without terminating
 *-----
 */
void  sndB(void)
{
    while( 1 )
        putc(CONSOLE, 'B');
}
```

¿Qué sucede al ejecutar el programa?

```
#include <xinu.h>

void    sndA(void), sndB(void);

/*-----
 * main  --  example of creating processes in Xinu
 *-----
 */
void    xsh_programita(void)
{
    resume( create(sndA, 1024, 20, "process 1", 0) );
    resume( create(sndB, 1024, 20, "process 2", 0) );
}

/*-----
 * sndA  --  repeatedly emit 'A' on the console without terminating
 *-----
 */
void    sndA(void)
{
    while( 1 )
        putc(CONSOLE, 'A');
}

/*-----
 * sndB  --  repeatedly emit 'B' on the console without terminating
 *-----
 */
void    sndB(void)
{
    while( 1 )
        putc(CONSOLE, 'B');
}
```

Lo que sucede es que los procesos se alternan menos seguido, generando cadenas largas de letras que corresponden al proceso sndA, luego al proceso sndB y viceversa. Es decir, tardan mucho más en conmutar de lo que se quisiera.

```
/* the run-time stack */
#define QUANTUM 2 /* time slice in milliseconds */
```

### Ejercicio 5. Finalización de procesos en Xinu.

Incorpore a main, luego de creado los procesos del Ejercicio 4, una espera de 10 segundos. Y que luego de la espera finalice los dos procesos iniciados anteriormente. (Ayuda: almacenar los PIDs de los procesos y averiguar cómo se llama la system call para finalizar procesos).

```
#include <xinu.h>

void  sndA(void), sndB(void);

//Variables globales para almacenar los PIDS de los procesos.
pid32 pidA, pidB; //pid32 es un tipo de datos entero de 32 bits que almacena el id único
de un proceso.

/*-----
 * main  --  example of creating processes in Xinu
 *-----
 */
void  xsh_programita(void)
{
    pidA =  create(sndA, 1024, 20, "process 1", 0) ;
    pidB =  create(sndB, 1024, 20, "process 2", 0) ;
    resume(pidA);
    resume(pidB);

    //Esperar 10seg
    sleep(10);
    //finalizar los procesos
    kill(pidA);
    kill(pidB);
}

/*-----
 * sndA  --  repeatedly emit 'A' on the console without terminating
 *-----
 */
void  sndA(void)
{
    while( 1 )
        putc(CONSOLE, 'A');
}

/*-----
 * sndB  --  repeatedly emit 'B' on the console without terminating
 *-----
 */
void  sndB(void)
{
    while( 1 )
        putc(CONSOLE, 'B');
}
```

### Ejercicio 6. Varios procesos reutilizando código ejecutable.

Incorpore a su programa código para crear los siguientes dos procesos que se observan en el siguiente ejemplo:

```
/* ex3.c - main, sndch */

#include <xinu.h>

void  sndch(char);

/*-----
 * main  --  example of 2 processes executing the same code concurrently
 *-----
 */
void  xsh_programita(void)
{

    /*
        FORMA DE CREAR UN PROCESO UN XINU
        resume (create(funcion, cantBytesPila, prioridad, nombreDelProceso,
cantArgumentos, Argumentos))
        funcion: es donde se comienza a ejecutar el programa

    */
    //resume( create(sndch, 1024, 20, "send A", 1, 'A') ); // el proceso A tiene 1024
bytes para su pila
    //resume( create(sndch, 1024, 20, "send B", 1, 'B') ); // el proceso B tiene 1024
bytes para su pila

    resume( create(sndch, 8192, 20, "send A", 1, 'A') );
    resume( create(sndch, 8192, 20, "send B", 1, 'B') );
}

/*-----
 * sndch  --  output a character on a serial device indefinitely
 *-----
 */
void  sndch(char ch)
{

    while ( 1 )
        putc(CONSOLE, ch);
}
```

### Ejercicio 7. Creación de procesos en Linux.

Utilice el ejemplo de la clase para crear un programa en Linux, que le pida al sistema operativo crear un nuevo proceso hijo. El proceso hijo debe obtener los números primos del 1000 al 5000. El proceso padre (luego de crear el hijo) debe obtener los números primos del 0 al 1000. Cuando finalice debe pedirle al kernel Linux que finalice el proceso hijo. Luego, debe pedirle al sistema operativo finalizar (con exit()).

(REVISAR)



```

/* ejercicio 7a */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

//Función para verificar si un número es primo.

int esPrimo(int num){
    int resultado = 1; //empieza siendo verdadero.
    if(num <= 1){ //Si es menor o igual a 1, no es primo.
        resultado = 0; //falso
    }else{
        for(int i = 2; i < num; i++){
            if(num % i == 0) //si es divisible por un número, ya no es primo.
                resultado = 0;
        }
    }
    return resultado;
}

//Funcion para obtener numeros primos en un rango

void obtenerPrimos(int inicio, int fin){
    printf("Numeros primos entre %d y %d:\n",inicio,fin);
    for(int i = inicio;i <=fin;i++){
        if(esPrimo(i)){
            printf("%d " ,i);
        }
    }
    printf("\n");
}

int main(){
    pid_t pid = fork(); //creación de proceso hijo.

    if(pid == 0){ //proceso hijo
        obtenerPrimos(1000,5000);
    }else{
        //proceso padre
        wait(NULL); //en espera
        obtenerPrimos(0,1000);

        //finalizar el proceso hijo
        printf("Finalizar el proceso hijo...\n");
        kill(pid,SIGTERM); //mata al hijo.
    }
}

```

```

}
printf("Finalizando el proceso padre...\n");
exit(EXIT_SUCCESS);

}

```

¿Cuál es el último número primo calculado por el proceso hijo?. ¿Le fue posible calcular todos los que debía calcular?

El último número primo calculado por el proceso hijo es el 4999. Fue posible calcular todos los que debía calcular.

En este caso, al tener un wait que espera hasta que termine el proceso hijo, es posible calcular todos los números primos del proceso hijo.

Sthey: El ultimo numero calculado por el proceso hijo es 4999. Si

### EJERCICIO 7-BIS

Utilice el ejemplo de la clase para crear un programa en Linux, que le pida al sistema operativo crear un nuevo proceso hijo. El nuevo proceso hijo debe reemplazar su imagen por el del programa /usr/bin/date. El padre debe finalizar realizando una llamada a exit(). ¿Qué funciones de C realizaron llamadas al sistema?

```

/* ejercicio 7bis TP1 */

#include <stdio.h>
#include <stdlib.h>

int main(){
    int pid;
    pid = fork();

    if( pid == 0) {
        /*proceso hijo que cambia su imagen*/
        printf("Soy el hijo\n");
        execv("/bin/date",NULL);
    }else{
        /*proceso padre*/
        //wait(0);
        printf("Soy el padre\n");
        exit(getpid());
    }
}

```

```

Soy el HIJO
paula.coronel@aula3-cpu6:~$ gcc TP1EJ7b.c -o ejecutable
paula.coronel@aula3-cpu6:~$ ./ejecutable
Soy el PADRE
Soy el HIJO
paula.coronel@aula3-cpu6:~$ jue 04 abr 2024 13:38:06 -03

```

### Ejercicio 8:

Douglas Comer en el libro que documenta la implementación de Xinu dice: The name stands for Xinu Is Not Unix. As we will see, the internal structure of Xinu differs completely from the internal structure of Unix (or

Linux). Xinu is smaller, more elegant, and easier to understand. Ahora compare las funciones que se utilizan en Linux y en Xinu para pedirle luego al sistema crear procesos. ¿Cuál es más fácil en su opinión?, ¿las funciones para realizar system calls en Linux o en Xinu? Si su respuesta fue Linux, entonces ¿por qué piensa que Douglas Comer diría que Xinu es más elegante y más fácil de entender?

En Linux, para solicitar al sistema la creación de un proceso, se utiliza principalmente la llamada al sistema **fork()** para cargar un nuevo programa en el proceso creado.

En Xinu, la creación de procesos se realiza utilizando la función **create()**. Esta función toma como argumentos el nombre del proceso, el tamaño de la pila del proceso, la prioridad del proceso y una lista variable de argumentos adicionales dependiendo de la implementación específica de Xinu. La función **create()** crea un nuevo proceso en el sistema Xinu.

- El primer argumento es una función en c (nombre del proceso)
- El segundo argumento es la cantidad de bytes que va a tener la pila
- El tercer argumento es la prioridad. (En Xinu a mayor valor mayor prioridad).
- El cuarto argumento es solamente el nombre (una cadena de ascii)
- Los argumentos restantes son los argumentos que le quieres pasar al nuevo proceso.

La manera más fácil a simple vista es la forma de crear procesos en Linux.

Pero en Xinu los system calls son más específicos para las operaciones que el sistema necesita realizar, esto hace que las funciones en Xinu sean más simples y fáciles de entender.

Douglas Comer dice que Xinu es más elegante y más fácil de entender a pesar de que las funciones en Linux puedan parecer más fáciles porque son más específicas y permite entender mejor el SO.

### Ejercicio 9:

Observar el archivo *include/process.h*. ¿Qué campos contiene la tabla de procesos en Xinu?. ¿Piensa que falta algo más que deba contemplar el sistema operativo para gestionar un proceso?

1) **prstate**: Indica el estado del proceso, que puede ser uno de los siguientes valores:

- **PR\_FREE**: La entrada de la tabla de procesos está sin usar.
- **PR\_CURR**: El proceso está actualmente en ejecución.
- **PR\_READY**: El proceso está listo para ejecutarse.
- **PR\_RECV**: El proceso está esperando un mensaje.
- **PR\_SLEEP**: El proceso está dormido.
- **PR\_SUSP**: El proceso está suspendido.
- **PR\_WAIT**: El proceso está esperando en una cola de semáforos.
- **PR\_RECTIM**: El proceso está recibiendo con un tiempo de espera.

2) **prprio**: Prioridad del proceso.

3) **prstkptr**: Puntero al inicio de la pila guardada del proceso.

4) **prstkbase**: Puntero al inicio de la pila en tiempo de ejecución del proceso.

5) **prstklen**: Longitud de la pila en bytes.

6) **prname**: Nombre del proceso.

7) **prsem**: Identificador del semáforo en el que el proceso está esperando.

8) **prparent**: ID del proceso creador.

9) **prmsg**: Mensaje enviado a este proceso.

10) **prhasmsg**: Indicador de si el mensaje es válido.

11) **prdesc[NDESC]**: Descriptores de dispositivo para el proceso.

Se define constantes relacionadas con la inicialización y la gestión de procesos, como la longitud del nombre del proceso (PNMLEN), el tamaño de la pila inicial (INITSTK), la prioridad inicial (INITPRIO),

### Ejercicio 10. Portar una aplicación entre sistemas operativos.

Portar el juego del ahorcado al sistema operativo Xinu. Agregue un acceso al shell para poder ejecutarlo.

¿Qué modificaciones fue necesario realizar para portar el juego?

AYUDA 0: system() es una función de la biblioteca en Linux, no se encuentra en Xinu. Averiguar sobre una equivalente.

AYUDA 1: si utilizó alguna otra función de biblioteca (permitida) entonces averiguar su equivalente en Xinu (si existe)

```
#include <xinu.h>

// juego del ahorcado.

int longitud(const char palabra[])
// método que calcula la longitud de una cadena de caracteres.
{
    int cantidad = 0;
    int direccion = 0;
    while (palabra[direccion] != '\0')
    {
        direccion++;
        cantidad++;
    }
    return cantidad;
}

int ahorcado()
{
    const int MAX_INTENTOS = 6; // guarda la cantidad de intentos
    posibles. No cambia.
    const char PALABRA_SECRETA[] = "ahorcado"; // Guarda la palabra a adivinar como
    un arreglo de caracteres.
    int intentos = 0; // Intentos actuales del juego
    char letra; // carácter actual puesto por el
    usuario.
    int palabra_len = longitud(PALABRA_SECRETA); // Cantidad de caracteres.
    char palabra_mostrada[palabra_len + 1]; // Palabra mostrada en pantalla.
    Arranca sin letras.
    int i; // contador.
    palabra_mostrada[palabra_len] = '\0';
    // Inicializar la palabra mostrada con guiones bajos
    for (i = 0; i < palabra_len; i++)
    {
```

```

    palabra_mostrada[i] = '_';
}
// Agregar el carácter nulo al final de la cadena

/* Decirle al sistema que el modo input es RAW */
control(CONSOLE, TC_MODER, 0, 0); /* coloca el input de la consola (tty) en modo
raw */

printf("\n¡Bienvenido al juego del Ahorcado!\n");
printf("Adivina la palabra secreta:\n");

while (1)
{
    printf("\nPalabra: %s\n", palabra_mostrada);
    printf("Intentos restantes: %d\n", MAX_INTENTOS - intentos);
    printf("\r                                     ");
    printf("ingrese una letra (0 para salir): ");
    letra = getchar();

    if (letra == '0') // Salir del juego si se ingresa '0'
        break;

    int acierto = 0; // controla si fue acierto o no. 0 = No hay acierto . 1 =
Hay acierto.
    for (i = 0; i < palabra_len; i++)
    {
        if (PALABRA_SECRETA[i] == letra)
        {
            // si el caracter de la palabra a adivinar
es igual a la letra que se ingresó...
            palabra_mostrada[i] = letra; // Se guarda en la palabra mostrada.
            acierto = 1; // Hay acierto.
        }
    }

    if (!acierto)
    { // Si no hay acierto, se consume un intento.
        intentos++;
        printf("\r                                     ");
        printf("La letra '%c' no está en la palabra.\n", letra);
    }

    // Verificar si se han agotado los intentos
    if (intentos >= MAX_INTENTOS)
    {
        printf("\n¡Perdiste! La palabra secreta era: %s\n", PALABRA_SECRETA);
    }
}

```

```

        break;
    }

    // Verificar si se ha adivinado la palabra
    int acertado = 1; // Controla si se acertó la palabra completa o no. 0 = No se
    adivinó toda la palabra secreta. 1 = Se adivinó toda la palabra secreta.
    for (i = 0; i < palabra_len; i++)
    {
        if (palabra_mostrada[i] != PALABRA_SECRETA[i])
        { // Si hay un carácter distinto, entonces todavia no se acertó.
            acertado = 0;
            break;
        }
    }
    if (acertado)
    { // Si se adivinó la palabra secreta, termina el juego.
        printf("\n¡Felicidades! ¡Adivinaste la palabra secreta %s!\n",
        PALABRA_SECRETA);
        break;
    }
}
control(CONSOLE, TC_MODEC, 0, 0); /* coloca el input de la consola (tty) en modo
cooked */
}

```

El equivalente a system() que si es apropiado dentro de Xinu es utilizar la función control(), la cual está definida en la carpeta "system".

```

#include <xinu.h>

int longitud(const char palabra[]) {
    int cantidad = 0;
    int direccion = 0;
    while (palabra[direccion] != '\0') {
        direccion++;
        cantidad++;
    }
    return cantidad;
}

int xsh_programa()
{
    const int CANT_INTENTOS = 6;
    const char cadenaSecreta[] = "PAULA CORONEL";
    const int longitudSecreta = longitud(cadenaSecreta);
    char cadenaMostrada[longitudSecreta + 1];
    int intentos = 0;
    char letra;

    for (int i = 0; i < longitudSecreta; i++)

```

```

{
    if (cadenaSecreta[i] == ' ')
    {
        cadenaMostrada[i] = ' ';
    }
    else
    {
        cadenaMostrada[i] = '_';
    }
}
cadenaMostrada[longitudSecreta] = '\0';
/* Decirle al sistema que el modo input es RAW */
// system("/bin/stty raw"); Linux no en Xinu

printf("\n¡Bienvenido al juego del Ahorcado!\n");
printf("Adivina la frase secreta:\n");

while (1)
{
    printf("\nFrase: %s\n", cadenaMostrada);
    printf("\nIntentos restantes: %d\n", (CANT_INTENTOS - intentos));
    printf("\r                                     ");
    printf("\r letra = %c  ingrese una letra (0 para salir): ", letra);
    //letra = getchar(); en linux
    letra = getc(stdin); // xinu

    // Limpiar el búfer de entrada(con el fin de evitar el salto de línea)
    while (letra == '\n' && letra != EOF) { //si es salto de línea, ingresara de
nuevo.
        letra = getc(stdin);
    }

    if (letra == '0') // Sale del juego si ingresa 0
        break;

    int exito = 0;
    for (int i = 0; i < longitudSecreta; i++)
    {
        if (cadenaSecreta[i] == letra)
        {
            exito = 1;
            cadenaMostrada[i] = letra;
        }
    }
    if (exito == 1)
    {
        printf("\n ACERTASTE!!!\n");
    }
    else
    {
        intentos++;
    }
}

```

```

        printf("\r                                     ");
        printf("La letra '%c' no está en la frase.\n", letra);
    }

    if (intentos >= CANT_INTENTOS)
    {
        printf("\n PERDISTE :C la frase secreta era: %s\n", cadenaSecreta);
        break;
    } else {
        int adivino = 1;
        int i = 0;
        while (adivino && i < longitudSecreta)
        {
            if (cadenaMostrada[i] != cadenaSecreta[i])
            {
                adivino = 0;
            }
            i++;
        }
        if (adivino)
        {
            printf("\n GANASTE!!! Adivinaste la frase secreta que es: %s\n",
cadenaSecreta);
            break;
        }
    }
}

// system("/bin/stty sane erase ^H"); Linux no en Xinu
}

```

## Trabajo Práctico 2

### EJERCICIO 1

#### a) Describir el planificador de procesos de Xinu.

El planificador de procesos es el componente del kernel que selecciona un proceso para asignarle la CPU. El planificador de procesos de Xinu implementa una política de planificación Round Robin para distribuir el tiempo de CPU entre los procesos de manera justa



- b) Implementar en Xinu un planificador de CPU de alto nivel, que define una política de planificación. La política es la siguiente: dado tres procesos a, b y c, el sistema operativo le asigne CPU como sigue: - a: 60% del tiempo - b: 30% del tiempo - c: 10% del tiempo

```
#include <xinu.h>

void incrementar (char ch){
    while(1){
        printf ("Proceso = %c \n", ch);
    }
}

void matar (char ch, int pid_a, int pid_b, int pid_c){
    printf ("Proceso = %cATADOR", ch);
    kill(pid_a);
    kill(pid_b);
    kill(pid_c);
}

void high_level_scheduler ( /* argumentos: */
    int pid_a, int ms_a, /* PID de a y rafaga de a en ms */
    int pid_b, int ms_b, /* PID de b y rafaga de b en ms */
    int pid_c, int ms_c, /* PID de c y rafaga de c en ms */
    int pid_m, int ms_m
){
    int prio_a, prio_b, prio_c, pid_plan, prio_plan, prio_aux;

    while (1) {
        /* planificar aquí el proceso a */

        planificarProceso(pid_a, ms_a);

        /* planificar aquí el proceso b */

        planificarProceso(pid_b, ms_b);

        /* planificar aquí el proceso c */

        planificarProceso(pid_c, ms_c);

        /* planificar aquí el proceso m */

        planificarProceso(pid_m, ms_m);
    }
}

void planificarProceso (int pid, int ms){

    int prio, pid_plan, prio_plan, prio_aux;
    /* obtener el PID del planificador (proceso actual) */
```

```

pid_plan = getpid();
/*obtener la prioridad del planificador (proceso con la mas alta prioridad)*/
prio_plan = getprio(pid_plan);
/* obtener la prioridad del proceso que vino por parámetro*/
prio = getprio(pid);

/* cambiar la prioridad del proceso, a un valor igual a la
   prioridad del planificador menos 1 */
prio_aux = prio;
chprio(pid, (prio_plan - 1));

/* liberar la CPU por ms_a ms (ponerse a dormir), por lo
   que Xinu asignará la CPU al proceso*/
sleepms(ms);

/* devolverle al proceso su prioridad original */
chprio(pid, prio_aux);
}

void xsh_planificador(){

    int duracion_rafaga, pid_a, ms_a, pid_b, ms_b, pid_c, ms_c, pid_m, ms_m, pid_plan;
    double n = 0;
    pid_a = create(incrementar, 1024, 20, "send A", 2, 'A', n);
    pid_b = create(incrementar, 1024, 15, "send B", 2, 'B', n);
    pid_c = create(incrementar, 1024, 10, "send C", 2, 'C', n);
    pid_m = create(matar, 1024, 1, "Matador", 4, 'M', pid_a, pid_b, pid_c );

    duracion_rafaga = 200; // 200 ms

    ms_a = duracion_rafaga * 0.6;
    ms_b = duracion_rafaga * 0.3;
    ms_c = duracion_rafaga * 0.1;
    ms_m = 500;

    pid_plan = create(high_level_scheduler, 1024, 30, "planificador", 8, pid_a, ms_a,
pid_b, ms_b, pid_c, ms_c, pid_m, ms_m);

    resume(pid_plan);
    resume(pid_a);
    resume(pid_b);
    resume(pid_c);
    resume(pid_m);

}

```

c. Como estamos ejecutando con Rond-Robins y este algoritmo es no bloqueante por lo tanto cuando ejecutamos un proceso lo hace por un tiempo y luego pasa al siguiente con mayor prioridad.

## EJERCICIO 2

- a. Agregar este programa al shell de Xinu (recuerde cambiar el nombre de main por el nombre que seleccione para el programa). Utilice una pila de 4KB para cada proceso:

```
#include <xinu.h>

void produce(void), consume(void);
    int32 n = 0; // Global variables are shared by all processes

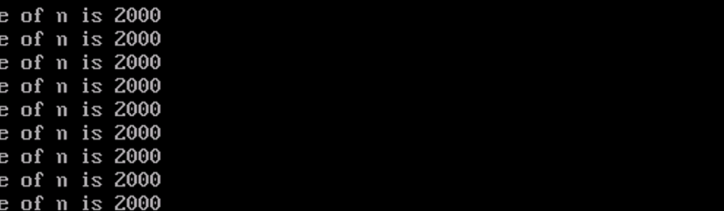
/*main Example of unsynchronized producer and consumer processes */

void ejercicio2tp2 (void){
    resume( create(consume, 4096, 20, "cons", 0) );
    resume( create(produce, 4096, 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 consume
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?



The screenshot shows a QEMU terminal window with a dark background and white text. The title bar at the top reads 'QEMU' and includes standard window control buttons (minimize, maximize, close). The terminal output consists of 20 identical lines, each stating 'The value of n is 2000'. The text is left-aligned and appears to be printed in a monospaced font. The lines are stacked vertically, filling most of the terminal's visible area.

El programa muestra por pantalla el valor de n repetidas veces, ya que tanto el productor como el consumidor realizan sus operaciones de forma secuencial sin sincronización ni coordinación entre ellos. El productor incrementa el valor de n en 2000 unidades, y luego el consumidor imprime este valor 2000 veces en su bucle.

- 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.

```
#include <xinu.h>
sid32 sem_produce, sem_consume;

void produce(void), consume(void);

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

/*main Example of unsynchronized producer and consumer processes */

void ejercicio2tp2 (void){

    sem_produce = semcreate(1);
    sem_consume = semcreate(0);

    resume( create(consume, 4096, 20, "cons", 0) );
    resume( create(produce, 4096, 20, "prod", 0) );
}

    // produce Increment n 2000 times and exit
void produce(void){
    int32 i;
    for( i=1; i<=2000 ; i++ ) {
        wait(sem_produce);
        n++;
        printf(" PRODUZCOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO %d \n", n);
        signal(sem_consume);
    }
}

    // consume - Print n 2000 times and exit consume
void consume(void){
    int32 i;
    for( i=1; i<=2000 ; i++ ){
        wait(sem_consume);
        printf("The value of n is %d \n", n);
        signal(sem_produce);
    }
}
```

```
QEMU
The value of n is 623
  PRODUCIENDO... 624
The value of n is 624
  PRODUCIENDO... 625
The value of n is 625
  PRODUCIENDO... 626
The value of n is 626
  PRODUCIENDO... 627
The value of n is 627
  PRODUCIENDO... 628
The value of n is 628
  PRODUCIENDO... 629
The value of n is 629
  PRODUCIENDO... 630
The value of n is 630
  PRODUCIENDO... 631
The value of n is 631
  PRODUCIENDO... 632
The value of n is 632
  PRODUCIENDO... 633
The value of n is 633
  PRODUCIENDO... 634
The value of n is 634
  PRODUCIENDO...
```

## EJERCICIO 3

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;
```

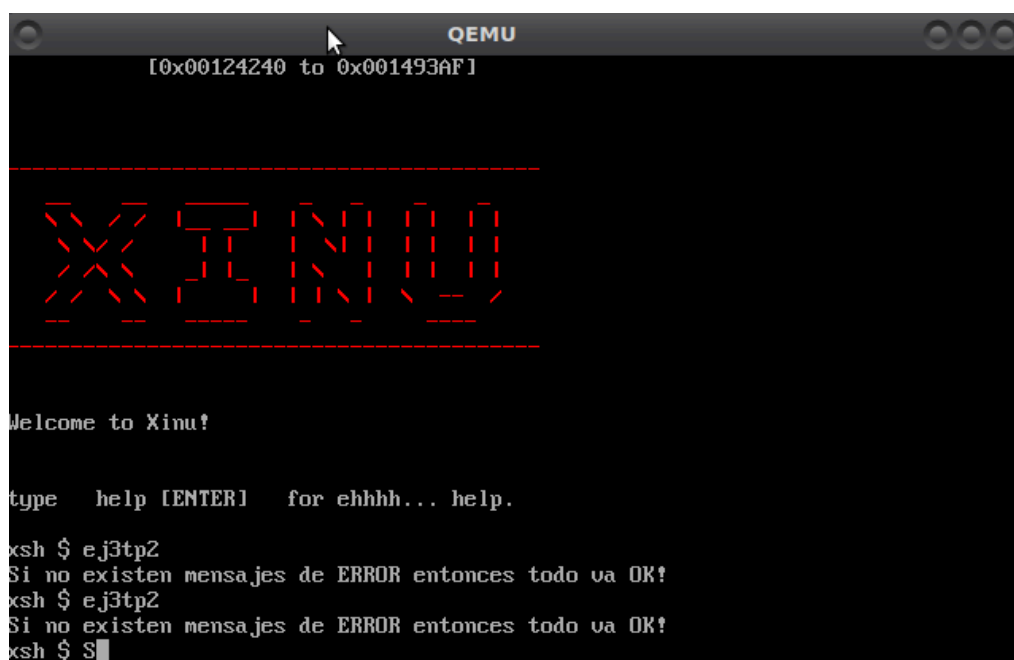
```

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;
}
}
}

```

**Modificar la cantidad de pila al crear los procesos. Utilice una pila de 8KB para cada proceso creado. Ejecutar el programa. ¿Por qué este programa (sus procesos) emiten mensajes de error? ¿Qué es posible hacer para solucionar el problema?**

El programa emite un mensaje de error porque al ingresar al primer if donde se cumple la condición de x sea múltiplo de 10 pero luego estando dentro del mismo if el valor de x se incrementa y ya no es un múltiplo de 10 y por eso sale por pantalla el mensaje de error. Lo que debería pasar es que x siga siendo múltiplo de 10 mientras esté dentro del primer if. Este error sucede porque la variable x es un recurso compartido entre dos procesos simultáneos pero no se está protegiendo correctamente, por esa razón se generan inconsistencia



```

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

void operar(void), incre(void);
unsigned char x = 0;
/*-----
 * mut -- programa con regiones criticas
 *-----
 */
void ejercicio3tp2(void) {
    resume( create(operar, 8192, 20, "process 1", 0) );
    resume( create(incre, 8192, 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 incre(void) {
    while (1) {
        x = x + 1;
    }
}

```

- 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()`;

Una posible implementación es utilizar un semáforo binario. Igualmente, HAY UNA DIFERENCIA IMPORTANTE ENTRE UN SEMÁFORO BINARIO Y UN MUTEX:

- En un semáforo binario, cualquier proceso puede realizar un `signal()` sobre el semáforo.
- En un mutex, el único proceso que puede realizar un `mutex_unlock()` es el proceso que realizó el `mutex_lock()`.

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

```
/* mut.c - mut, operar, incrementar */
#include <xinu.h>
sid32 semaforo;

void operar(void), incre(void);
unsigned char x = 0;
int pid = 0;
/*-----
 * mut -- programa con regiones criticas
 *-----
 */
void ejercicio3tp2(void) {
    mutex_init();
    resume( create(operar, 8192, 20, "process 1", 0) );
    resume( create(incre, 8192, 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 */
        mutex_lock();
        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);
            }
        }
        mutex_unlock();
    }
}
/*-----
 * incrementar x
 *-----
```



```

*/

void incre(void) {
    while (1) {
        mutex_lock();
        x = x + 1;
        mutex_unlock();
    }
}

void mutex_init(){
    semaforo = semcreate(1);
}

void mutex_lock(){
    wait(semaforo);
    pid = getpid();
}

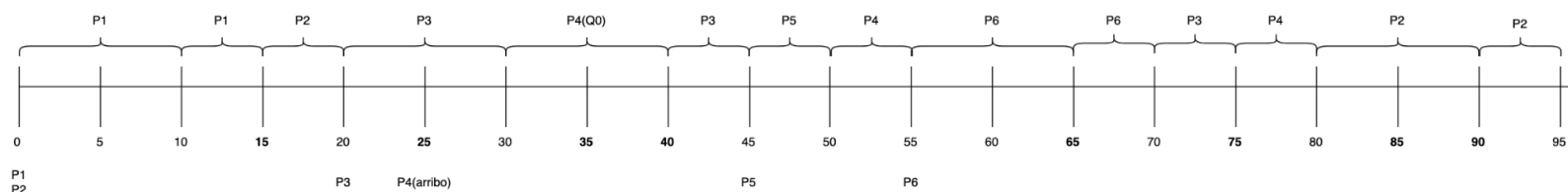
void mutex_unlock(){
    if(getpid() == pid){
        signal(semaforo);
    }
}

```

## EJERCICIO 4: Ejemplos de Planificación de CPU

4)a) [drawio](#)

### APROPIATIVO



b) ¿Cuánto es el tiempo de turnaround para cada proceso? ¿Y el valor medio contando todos los procesos?

Tiempo de turnaround para cada proceso:

P1 = 15;      P2 = 95;      P3 = 55;      P4 = 55;      P5 = 5; P6 = 20

Tiempo de turnaround promedio:  $(15+95+55+55+5+20) / 6 = 40.84$

c) ¿Cuál es el tiempo de espera para cada proceso? ¿Y el valor medio contando todos los procesos?

### Tiempo de espera:

P1 = 0;      P2 = 75;      P3 = 5;    P4 = 40;      P5 = 0;    P6 = 5

**Tiempo de espera promedio:**  $(0+75+5+40+0+5) / 6 = 20.833$

Sea un sistema con la siguiente carga de procesos. El planificador de CPU del sistema es por prioridades, round-robin, apropiativo.

| Proceso | Prioridad | Ráfaga | Arribo a la cola de listos |
|---------|-----------|--------|----------------------------|
| P1      | 8         | 15     | 0                          |
| P2      | 3         | 20     | 0                          |
| P3      | 4         | 20     | 20                         |
| P4      | 4         | 20     | 25                         |
| P5      | 5         | 5      | 45                         |
| P6      | 5         | 15     | 55                         |

A más alto número de prioridad mayor prioridad. El quantum del sistema es de 10 unidades. A misma prioridad el planificador es round-robin.

- a) Mostrar el orden de ejecución y las ráfagas de CPU de los procesos.

## EJERCICIO 5

Sea un sistema con la siguiente carga de procesos.

| Proceso | Ráfaga | Prioridad |
|---------|--------|-----------|
| P1      | 5      | 4         |
| P2      | 3      | 1         |
| P3      | 1      | 2         |
| P4      | 7      | 2         |
| P5      | 4      | 3         |

Los procesos arriban al sistema en el momento 0, en este orden: P1 , P2 , P3 , P4 , P5.

- a) Mostrar el orden de ejecución y las ráfagas de CPU de los procesos si el planificador es FCFS, SJF, con prioridades no-apropiatio, round-robin (quantum = 2)

### FCFS (Por orden de llegada, apropiativo)

|    |    |    |    |    |
|----|----|----|----|----|
| P1 | P2 | P3 | P4 | P5 |
| 0  | 5  | 8  | 9  | 16 |
|    |    |    |    | 20 |

### SJF(El trabajo mas corto primero, NO ES APROPIATIVO)

|    |    |    |    |    |
|----|----|----|----|----|
| P3 | P2 | P5 | P1 | P4 |
| 0  | 1  | 4  | 8  | 13 |
|    |    |    |    | 20 |

### Con prioridades no-apropiatio

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| P1 | P5 | P3 | P4 | P2 |    |
| 0  | 5  | 9  | 10 | 17 | 20 |

Round-Robin (quantum = 2) (Cada proceso se ejecuta un quantum. Se lo desaloja y se lo coloca al final de la cola de listos. APROPIATIVO. Ejecuta por orden de llegada)

|    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|
| P1 | P2 | P3 | P4 | P5 | P1 | P2 | P4 | P5 | P1 | P4 | P4 |    |
| 0  | 2  | 4  | 5  | 7  | 9  | 11 | 12 | 14 | 16 | 17 | 19 | 20 |

- b) ¿Cuánto es el tiempo de turnaround para cada proceso? ¿Y el valor medio contando todos los procesos? (para cada algoritmo).

#### FCFS

- Tiempo de turnaround para cada proceso:

$$P1 = 5 \quad P2 = 8 \quad P3 = 9 \quad P4 = 16 \quad P5 = 20$$

- Tiempo de turnaround promedio:  $(5 + 8 + 9 + 16 + 20) / 5 = 11.6$

#### SJF

- Tiempo de turnaround para cada proceso:

$$P1 = 13 \quad P2 = 4 \quad P3 = 1 \quad P4 = 20 \quad P5 = 8$$

- Tiempo de turnaround promedio:  $(13 + 4 + 1 + 20 + 8) / 5 = 9.2$

#### Con prioridades no-apropiatio

- Tiempo de turnaround para cada proceso:

$$P1 = 5 \quad P2 = 20 \quad P3 = 10 \quad P4 = 17 \quad P5 = 9$$

- Tiempo de turnaround promedio:  $(5 + 20 + 10 + 17 + 9) / 5 = 12.2$

#### Round-Robin (quantum = 2)

- Tiempo de turnaround para cada proceso:

$$P1 = 17 \quad P2 = 12 \quad P3 = 5 \quad P4 = 20 \quad P5 = 16$$

- Tiempo de turnaround promedio:  $(17 + 12 + 5 + 20 + 16) / 5 = 14$

- c) ¿Cuál es el tiempo de espera para cada proceso? ¿Y el valor medio contando todos los procesos? (para cada algoritmo).

#### FCFS

- Tiempo de espera:

$$P1 = 0 \quad P2 = 5 \quad P3 = 8 \quad P4 = 9 \quad P5 = 16$$

- Tiempo de espera promedio:  $(0 + 5 + 8 + 9 + 16) / 5 = 7.6$

#### SJF

- Tiempo de espera:

P1 = 8          P2 = 1          P3 = 0          P4 = 13          P5 = 4

- **Tiempo de espera promedio:**  $(8+1+0+13+4) / 5 = 5.2$

#### Con prioridades no-apropiadas

- **Tiempo de espera:**

P1 = 0          P2 = 17          P3 = 9          P4 = 10          P5 = 5

- **Tiempo de espera promedio:**  $(0+17+9+10+5) / 5 = 8.2$

#### Round-Robin (quantum = 2)

- **Tiempo de espera:**

→  $P1 = 0 + (9-2) + (16 - 11) = 12$

→  $P2 = 2 + (11-4) = 9$

→  $P3 = 4$

→  $P4 = 5 + (12 - 7) + (17-14) = 13$

→  $P5 = 7 + (14 - 9) = 12$

- **Tiempo de espera promedio:**  $(12+9+4+13+12) / 5 = 10$

d) ¿Cuál de los algoritmos produce el mínimo promedio de tiempo de espera?

El algoritmo de SJF (Shortest Job First) produce el menor promedio de tiempo de espera

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <sys/types.h>
```

```
#include <sys/mman.h>
```

```
#include <fcntl.h>
```

```
#include <string.h>
```

```
#define SHARED_MEM_SIZE 10000 // Tamaño de la región de memoria compartida
```

```
int main() {
```

```
    // Crear una región de memoria compartida
```

```
    int shm_fd = shm_open("/mi_memoria_compartida", O_CREAT | O_RDWR, 0666);
```

```
    if (shm_fd == -1) {
```

```
        perror("shm_open");
```

```
        exit(1);
```

```
    }
```

```

// Configurar el tamaño de la región de memoria compartida

if (ftruncate(shm_fd, SHARED_MEM_SIZE) == -1) {
    perror("ftruncate");
    exit(1);
}

// Mapear la región de memoria compartida

char *shared_memory = mmap(NULL, SHARED_MEM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);

if (shared_memory == MAP_FAILED) {
    perror("mmap");
    exit(1);
}

// Abrir el archivo de texto

FILE *file = fopen("/usr/share/doc/aufs-dkms/filesystems/aufs/design/06mmap.txt", "r");

if (file == NULL) {
    perror("fopen");
    exit(1);
}

// Leer el contenido del archivo y colocarlo en la región de memoria compartida

size_t bytesRead = fread(shared_memory, 1, SHARED_MEM_SIZE, file);

if (bytesRead == 0) {
    perror("fread");
    exit(1);
}

printf("Se leyeron %zu bytes del archivo y se colocaron en la memoria compartida.\n", bytesRead);

// Cerrar el archivo y liberar la memoria compartida

```

```

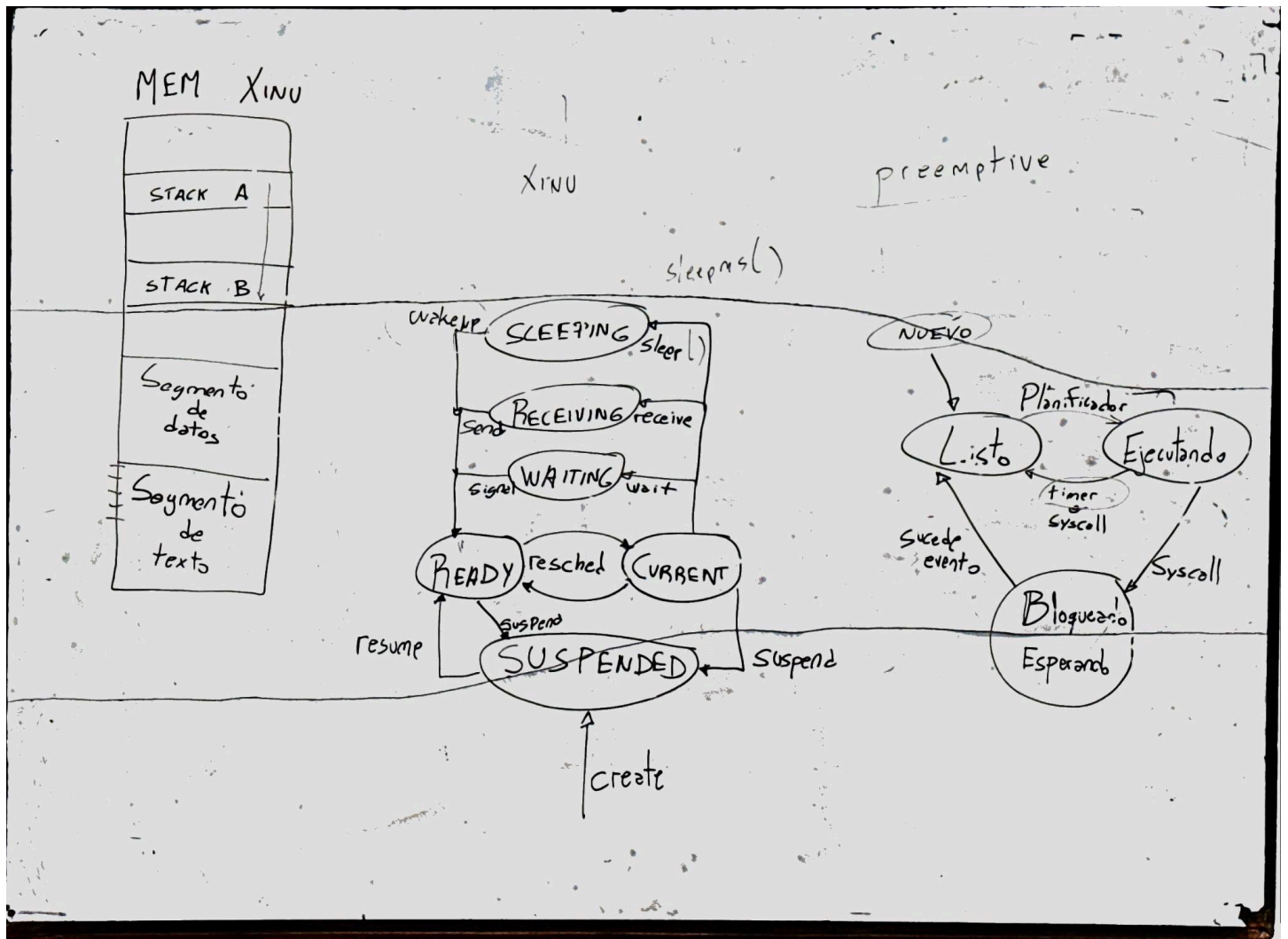
fclose(file);

munmap(shared_memory, SHARED_MEM_SIZE);

close(shm_fd);

return 0;
}

```



## /Trabajo Práctico N°3

Ejercicio 1. 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.

```

#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h> /* Para ftruncate */

```

```
#include <fcntl.h> /* Para constantes O_* */

const int SIZE = 4096;
const char *name = "compartido";
const char *pathName = "/usr/share/doc/powermgmt-base/power_supply.txt";

int fileDescriptor;
void *ptr;

int main()
{
    int buffSize = 1024;
    char buff[buffSize];

    /* Solicitar crear segmento de memoria compartida, escritura/lectura */
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    /* Configurar el tamaño del segmento compartido */
    ftruncate(shm_fd, SIZE);

    /* Mapear segmento de memoria compartida en el espacio de dirección del proceso */
    ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED)
    {
        printf("Mapeo fallido\n");
        return -1;
    }

    /* Abrimos el archivo solo para lectura */
    fileDescriptor = open(pathName, O_RDONLY);

    /* Copiamos el contenido del txt en buff */
    read(fileDescriptor, buff, buffSize);

    /* Copiamos el contenido de buff en el segmento de memoria compartida */
    sprintf(ptr, "%s", buff);

    /* Cerramos el archivo */
    if (close(fileDescriptor) == -1)
    {
        printf("Error al cerrar el archivo");
    }

    printf("Funciono C:");

    return 0;
}
```

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).

```
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h> /* Para ftruncate */
#include <fcntl.h> /* Para constantes O_* */
#include <stdlib.h>

const int SIZE = 4096;
const char *name = "compartido";

int fileDescriptor;
void *ptr;

int main()
{
    int buffSize = 1024;
    char buff[buffSize];

    /* Solicitar crear segmento de memoria compartida, escritura/lectura */
    int shm_fd = shm_open(name, O_RDONLY , 0666);

    /* Configurar el tamaño del segmento compartido */
    ftruncate(shm_fd, SIZE);

    /* Mapear segmento de memoria compartida en el espacio de dirección del proceso */
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED)
    {
        printf("Mapeo fallido\n");
        return -1;
    }

    /* Leemos de la memoria compartida el archivo que contiene */
    printf("%s", (char *)ptr);

    /* Remueve el segmento de memoria compartida */
    if (shm_unlink(name) == -1) {
        printf("Error removing %s\n", name);
        exit(-1);
    }

    return 0;
}
```

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

Lo hicimos en el b



Ejercicio 2. 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.

```
#include "titlescreen.h"
#include "playerImage.h"
#include "enemy.h"
#include "boss.h"
#include "gameover.h"
#include "shoot.h"
#include <xinu.h>

extern unsigned char tecla_actual;
typedef unsigned short u16;
#define RGB(r, g, b) (r | (g << 5) | (b << 10))
// #define REG_DISPCNT *(u16 *)0x4000000
#define extern videoBuffer
#define MODE3 3
#define BG2_ENABLE (1 << 10)
#define WHITE RGB(31, 31, 31)
#define BLACK RGB(0, 0, 0)

/*
#define BUTTON_A (1<<0)
```

```

#define BUTTON_B      (1<<1)
#define BUTTON_SELECT (1<<2)
#define BUTTON_START  (1<<3)
#define BUTTON_RIGHT  (1<<4)
#define BUTTON_LEFT   (1<<5)
#define BUTTON_UP      (1<<6)
#define BUTTON_DOWN    (1<<7)
#define BUTTON_R       (1<<8)
#define BUTTON_L       (1<<9)
#define KEY_DOWN_NOW(key)  (~(BUTTONS) & key)
*/
//#define BUTTONS *(volatile unsigned int *)0x4000130

#define BUTTON_A      0x24
#define BUTTON_B      0x25
#define BUTTON_SELECT  0x03
#define BUTTON_START   0x2c
#define BUTTON_RIGHT   0x1f
#define BUTTON_LEFT    0x1e
#define BUTTON_UP      'w'
#define BUTTON_DOWN    's'
#define BUTTON_R       '1'
#define BUTTON_L       '2'
#define BUTTON_FIN     0x2d
#define KEY_DOWN_NOW(key)  (tecla_actual == key)

//variable definitions
#define playerspeed 2
#define enemyspeed 1
#define fastXSpeed 3
#define fastYSpeed 2

void setPixel(int x, int y, u16 color);
void drawRect(int x, int y, int width, int height, u16 color);
void drawHollowRect(int x, int y, int width, int height, u16 color);
void drawImage3(int x, int y, int width, int height, const u16* image);
void delay_galaga();
void waitForVBlank();

//helpers
void initialize();
void drawEnemies();
void endGame();
int controlJuego();
int vidas_puntaje();
int juego();
int collision(u16 enemyX, u16 enemyY, u16 enemyWidth, u16 enemyHeight, u16 playerX, u16 playerY);
//variables globales agregadas (los pid de los procesos)
int pid_p1, pid_p2, pid_control;
//variable global del valor de puntos por cada disparo

```

```

int puntos = 5;

//objects
struct Players {
    volatile u16 playerX;
    volatile u16 playerY;
};
struct Enemy {
    volatile u16 enemyX;
    volatile u16 enemyY;
};
struct FastEnemy {
    volatile u16 fastX;
    volatile u16 fastY;
};

int shoots[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int curr_shot = 0;
#define N_SHOOTS 10

//PROCESO CONTROL
int controlJuego(void){
    int msg;
    //el proceso control crea a los dos procesos
    pid_p1 = create(juego,1024,20,"proceso 1",0);
    pid_p2 = create(vidas_puntaje,1024,20,"proceso 2",0);
    resume(pid_p1);
    resume(pid_p2);
    //espera por un mensaje de finalización del proceso 1
    msg = receive();
    kill(pid_p1);
    kill(pid_p2);
    endGame();
}

//PROCESO 2

int vidas_puntaje(void){
    int mensaje;
    char buffer[80];
    int cantVidas, cantPuntaje;
    cantVidas = 3;
    cantPuntaje = 0;
    //muestra las vidas que le quedan al jugador y su puntaje
    while (1){
        sprintf(buffer,"Puntaje: %d          Vida: %d", cantPuntaje, cantVidas);
        print_text_on_vga(10, 240, buffer);
        mensaje = receive();//queda esperando a que el proceso 1 le diga si gano puntos
        o perdió una vida
        if(mensaje > 1){ //si es mayor a uno, significa que ganó puntos
            //gano puntos
            cantPuntaje = cantPuntaje + mensaje;

```

```

    }else{
        //perdio vida
        if(mensaje ==1){ //si es igual a uno, perdió una vida
            cantVidas = cantVidas - 1;
            if(cantVidas == 0){ //si perdió todas las vidas:
                sprintf(buffer,"Puntaje: %d          Vida: %d", cantPuntaje,
cantVidas);

                print_text_on_vga(10, 240, buffer);
                send(pid_control,1); //manda un mensaje al proceso control
para terminar el juego
            }
        }
    }
}

```

```

//PROCESO 1
int juego(void) {
    //int msg,mensaje;
    //easy enemy wave set setup
    struct Enemy easyEnemies[9];
    for (int a = 0; a < 9; a++) {
        easyEnemies[a].enemyX = (28*a);
        easyEnemies[a].enemyY = 0;
    }
    easyEnemies[1].enemyX = 240;
    easyEnemies[4].enemyX = 240;
    easyEnemies[8].enemyX = 240;
    //difficult enemies setup
    struct Enemy hardEnemies[9];
    for (int a = 0; a < 9; a++) {
        hardEnemies[a].enemyX = (28*a);
        hardEnemies[a].enemyY = 160;
    }
    hardEnemies[3].enemyX = 240;
    hardEnemies[6].enemyX = 240;
    //player setup
    struct Players player;
    player.playerX = 120;
    player.playerY = 136;
    //fast enemy "boss" setup
    struct FastEnemy fast;
    fast.fastX = 0;
    fast.fastY = 30;

    // REG_DISPCNT = MODE3 | BG2_ENABLE;
    //inititalize title screen
    print_text_on_vga(10, 20, "GALAGA ");
    drawImage3(0, 0, 240, 160, titlescreen);
    while(1) {
        if (KEY_DOWN_NOW(BUTTON_START)) {
            break;
        }
    }
}

```

```

    }
}
//start black screen for drawing
for (int i = 0; i < 240; i++) {
    for (int j = 0; j < 160; j++) {
        setPixel(i, j, BLACK);
    }
}
while(1) {
    //go back to title screen if select button is pressed
    if (KEY_DOWN_NOW(BUTTON_SELECT)) {
        //initialize();
        juego();
    }
    //avisa al proceso control que finalice el juego si el usuario selecciona la
tecla correspondiente
    if (KEY_DOWN_NOW(BUTTON_FIN)) {
        send(pid_control,1);
    }

    //player shots
    if (KEY_DOWN_NOW(BUTTON_A)) {
        if (shoots[curr_shot] == 0) {
            shoots[curr_shot] = 136*240 + player.playerX+9; /* 24 width player
*/
            curr_shot++;
            if (curr_shot >= N_SHOOTS)
                curr_shot = 0;
        };
    }
    //player movement input
    if (KEY_DOWN_NOW(BUTTON_LEFT) && (player.playerX > 0)) {
        player.playerX -= playerspeed;
    }
    if (KEY_DOWN_NOW(BUTTON_RIGHT) && (player.playerX <= 216)) {
        player.playerX += playerspeed;
    }
    if (KEY_DOWN_NOW(BUTTON_UP) && (player.playerY > 25)) {
        player.playerY -= playerspeed;
    }
    if (KEY_DOWN_NOW(BUTTON_DOWN) && (player.playerY <= 136)) {
        player.playerY += playerspeed;
    }
    waitForVBlank();
    sleepms(50);
    //draw player
    drawImage3(player.playerX, player.playerY, 24, 24, playerImage);
    drawHollowRect(player.playerX - 1, player.playerY - 1, 26, 26, BLACK);
    drawHollowRect(player.playerX - 2, player.playerY - 2, 28, 28, BLACK);
    //draw easy enemies with downward movement
    for (int a = 0; a < 9; a++) {
        easyEnemies[a].enemyY += enemyspeed;
    }
}

```

```

        drawImage3(easyEnemies[a].enemyX, easyEnemies[a].enemyY, 20, 20, enemy);
        if (collision(easyEnemies[a].enemyX, easyEnemies[a].enemyY, 20, 20,
player.playerX, player.playerY)) {
            //se produjo una colisión, pierde una vida y se avisa al proceso 2
            //endGame();
            drawRect(easyEnemies[a].enemyX, easyEnemies[a].enemyY, 20, 20,
BLACK);

            easyEnemies[a].enemyY = 0;
            send(pid_p2, 1);
        }
        if (easyEnemies[a].enemyY >= 160) {
            easyEnemies[a].enemyY = 0;
        }
    }

    //draw shots
    for (int i = 0; i < N_SHOOTS; i++) {
        if (shoots[i] != 0) {
            drawRect((shoots[i] % 240), (shoots[i] / 240)+4, 5, 5, BLACK);
            drawImage3((shoots[i] % 240), (shoots[i] / 240), 5, 5, shoot);
            shoots[i] = shoots[i]-240*4;
            if (shoots[i] <=0)    shoots[i]=0;
        }

        // check hits of shoots
        for (int j = 0; j < 9; j++) {
            if ( (shoots[i] != 0) && collision(easyEnemies[j].enemyX,
easyEnemies[j].enemyY, 15, 15, shoots[i] % 240, shoots[i] / 240)) {
                drawRect(easyEnemies[j].enemyX, easyEnemies[j].enemyY, 20,
20, BLACK);

                drawRect((shoots[i] % 240), (shoots[i] / 240)+4, 5, 5, BLACK);
                //drawImage3((shoots[i] % 240), (shoots[i] / 240), 5, 5,
BLACK);

                easyEnemies[j].enemyY = 0;
                shoots[i] = 0;
                //el jugador logro dispararle al enemigo, avisa al proceso 2
que sume puntos
                send(pid_p2, puntos);
            }
        }
    }

    //draw hard enemies
    for (int a = 0; a < 9; a++) {
        hardEnemies[a].enemyY += enemyspeed;
        drawImage3(hardEnemies[a].enemyX, hardEnemies[a].enemyY, 20, 20, enemy);
        if (collision(hardEnemies[a].enemyX, hardEnemies[a].enemyY, 20, 20,
player.playerX, player.playerY)) {
            //se produjo una colisión, pierde una vida y se avisa al proceso 2
            //endGame();
            drawRect(hardEnemies[a].enemyX, hardEnemies[a].enemyY, 20, 20,
BLACK);

```

```

        easyEnemies[a].enemyY = 0;
        send(pid_p2, 1);
    }
    if (hardEnemies[a].enemyY >= 228) {
        hardEnemies[a].enemyY = 0;
    }
    if ((hardEnemies[a].enemyY >= 200) && (easyEnemies[a].enemyY <=45)) {
        hardEnemies[a].enemyY = 160;
    }
    //space enemies apart
    if ((hardEnemies[a].enemyY >= 200) && (easyEnemies[a].enemyY <=45)) {
        hardEnemies[a].enemyY = 160;
    }
    if ((easyEnemies[a].enemyY >= 120) && (hardEnemies[a].enemyY >=170)) {
        hardEnemies[a].enemyY = 160;
    }
}
//draw fast enemy
drawImage3(fast.fastX, fast.fastY, 15, 15, boss);
drawHollowRect(fast.fastX - 1, fast.fastY - 1, 17, 17, BLACK);
drawHollowRect(fast.fastX - 2, fast.fastY - 2, 19, 19, BLACK);
if(collision(fast.fastX, fast.fastY, 15, 15, player.playerX, player.playerY)) {
    //se produjo una colisión, pierde una vida y se avisa al proceso 2
    //endGame();
    drawRect(fast.fastX, fast.fastY, 20, 20, BLACK);
    fast.fastX = 0;
    //fast.fastY = player.playerY - 20;
    send(pid_p2, 1);
}
//comentado por rafa
fast.fastX += fastXSpeed;
fast.fastY += fastYSpeed;
if (fast.fastX >= 240) {
    fast.fastX = 0;
}
if (fast.fastY >= 200) {
    fast.fastY = player.playerY - 20;
}
}
return 0;
}

```

```

int collision(u16 enemyX, u16 enemyY, u16 enemyWidth, u16 enemyHeight, u16 playerX, u16
playerY) {
    //check if bottom right corner of enemy is within player
    if (((enemyX + enemyWidth) > playerX) && ( (enemyY + enemyHeight)
    > playerY ) && ((enemyX + enemyWidth) < (playerX + 24))
    && ((enemyY + enemyWidth) < (playerY + 24))) {
        return 1;
    }
    //check bottom left corner of enemy

```

```

    if ( (enemyX < (playerX + 24))
        && (enemyX > playerX)
        && ((enemyY + enemyHeight) > playerY)
        && ((enemyY + enemyHeight) < (playerY + 24))
        ) {
        return 1;
    }
    //check top left corner
    if ( (enemyX < (playerX + 24))
        && (enemyX > playerX)
        && (enemyY > playerY)
        && (enemyY < (playerY + 24))
        ) {
        return 1;
    }
    //check top right corner
    if ( ((enemyX + enemyWidth) < (playerX + 24))
        && ((enemyX + enemyWidth) > playerX)
        && (enemyY > playerY)
        && (enemyY < (playerY + 24))
        ) {
        return 1;
    }
    return 0;
}

void endGame() {
    //start Game Over State
    drawImage3(0, 0, 240, 160, gameover);
    drawHollowRect(0, 0, 240, 160, WHITE);
    while(1) {
        if (KEY_DOWN_NOW(BUTTON_SELECT)) {
            galaga();
            break;
        }
        if (KEY_DOWN_NOW(BUTTON_START)) {
            galaga();
            break;
        }
    }
}

//CREACIÓN DE LOS PROCESOS

int galaga(void){
    pid_control = create(controlJuego,1024,20,"proceso control",0);
    resume(pid_control);
    sleep(2);
}

```



el mismo ejercicio solo que le agregué un puntMax que muestra el máximo puntaje obtenido en todas las partidas

```
#include "titlescreen.h"
#include "playerImage.h"
#include "enemy.h"
#include "boss.h"
#include "gameover.h"
#include "shoot.h"
#include <xinu.h>

extern unsigned char tecla_actual;
typedef unsigned short u16;
#define RGB(r, g, b) (r | (g << 5) | (b << 10))
// #define REG_DISPCNT *(u16 *)0x4000000
#define extern videoBuffer
#define MODE3 3
#define BG2_ENABLE (1 << 10)
#define WHITE RGB(31, 31, 31)
#define BLACK RGB(0, 0, 0)

/*
#define BUTTON_A      (1<<0)
#define BUTTON_B      (1<<1)
#define BUTTON_SELECT (1<<2)
#define BUTTON_START  (1<<3)
#define BUTTON_RIGHT   (1<<4)
#define BUTTON_LEFT    (1<<5)
#define BUTTON_UP      (1<<6)
#define BUTTON_DOWN    (1<<7)
#define BUTTON_R       (1<<8)
#define BUTTON_L       (1<<9)
#define KEY_DOWN_NOW(key)  (~(BUTTONS) & key)
*/
// #define BUTTONS *(volatile unsigned int *)0x4000130

#define BUTTON_A      0x24
#define BUTTON_B      0x25
#define BUTTON_SELECT 0x03
#define BUTTON_START  0x2c
#define BUTTON_RIGHT   0x1f
#define BUTTON_LEFT    0x1e
#define BUTTON_UP      'w'
#define BUTTON_DOWN    's'
#define BUTTON_R       '1'
#define BUTTON_L       '2'
#define BUTTON_FIN     0x2d //tecla x
```

```
#define KEY_DOWN_NOW(key)  (tecla_actual == key)

//variable definitions
#define playerspeed 2
#define enemyspeed 1
#define fastXSpeed 3
#define fastYSpeed 2

void setPixel(int x, int y, u16 color);
void drawRect(int x, int y, int width, int height, u16 color);
void drawHollowRect(int x, int y, int width, int height, u16 color);
void drawImage3(int x, int y, int width, int height, const u16* image);
void delay_galaga();
void waitForVBlank();

//helpers
void initialize();
void drawEnemies();
void endGame();
int controlJuego();
int vidas_puntaje();
int juego();
int collision(u16 enemyX, u16 enemyY, u16 enemyWidth, u16 enemyHeight, u16 playerX, u16 playerY);
//variables globales agregadas (los pid de los procesos)
int pid_p1, pid_p2, pid_control;
//variable global del valor de puntos por cada disparo
int puntos = 5;
//variable global del puntaje máximo.
int puntajeMax = 0;
//variables globales de la cantidad de vidas y la cantidad de puntaje iniciales.
int cantVidas = 3;
int cantPuntaje = 0;

//objects
struct Players {
    volatile u16 playerX;
    volatile u16 playerY;
};
struct Enemy {
    volatile u16 enemyX;
    volatile u16 enemyY;
};
struct FastEnemy {
    volatile u16 fastX;
    volatile u16 fastY;
};

int shoots[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int curr_shot = 0;
#define N_SHOOTS 10
```

```

//PROCESO CONTROL
int controlJuego(void){
    int msg;
    char buffer[80]; //guarda en la pantalla el puntaje máximo
    //el proceso control crea a los dos procesos

    pid_p1 = create(juego,1024,20,"proceso 1",0);
    pid_p2 = create(vidas_puntaje,1024,20,"proceso 2",0);
    resume(pid_p1);
    resume(pid_p2);
    //espera por un mensaje de finalización del proceso 1
    msg = receive();
    if(msg > puntajeMax){ //si hay un nuevo puntaje máximo
        puntajeMax = msg; //se setea el nuevo puntaje máximo
        sprintf(buffer,"Puntaje maximo: %d",puntajeMax);
        print_text_on_vga(10, 280, buffer);
    }

    kill(pid_p1);
    kill(pid_p2);
    endGame(); //si el puntaje obtenido no supera al máximo, solo termina el juego.
}

//PROCESO 2 (vidas y puntaje)
int vidas_puntaje(void){
    int mensaje;
    char buffer[80]; //guarda en la pantalla el puntaje actual
    char bufferVida[80]; //guarda en la pantalla la vida actual
    cantVidas = 3;
    cantPuntaje = 0;
    //muestra las vidas que le quedan al jugador y su puntaje
    sprintf(buffer, " "); //limpio el buffer
para evitar que se haga sobre lo anterior al poner el nuevo puntaje
    print_text_on_vga(10,260,buffer);
    while (1){
        sprintf(buffer,"Puntaje: %d", cantPuntaje);
        sprintf(bufferVida,"Vida: %d", cantVidas);
        print_text_on_vga(10, 260, buffer);
        print_text_on_vga(10,240,bufferVida);
        mensaje = receive();//queda esperando a que el proceso 1 le diga si gano puntos
o perdió una vida
        if(mensaje > 1){ //si es mayor a uno, significa que ganó puntos
            //gano puntos
            cantPuntaje = cantPuntaje + mensaje;
        }else{
            //perdio vida

```

```

        if(mensaje ==1){ //si es igual a uno, perdió una vida
            cantVidas = cantVidas - 1;
            if(cantVidas == 0){ //si perdió todas las vidas:
                sprintf(bufferVida,"Vida:  %d", cantVidas); //actualizo el buffer
de las vidas para que me muestre cuando pierde todas.
                print_text_on_vga(10,240,bufferVida);
                send(pid_control,cantPuntaje); //manda un mensaje con el
puntaje al proceso control para terminar el juego
            }
        }
    }
}

```

//PROCESO 1 (Galaga)

```

int juego(void) {

    //easy enemy wave set setup
    struct Enemy easyEnemies[9];
    for (int a = 0; a < 9; a++) {
        easyEnemies[a].enemyX = (28*a);
        easyEnemies[a].enemyY = 0;
    }
    easyEnemies[1].enemyX = 240;
    easyEnemies[4].enemyX = 240;
    easyEnemies[8].enemyX = 240;

    //difficult enemies setup
    struct Enemy hardEnemies[9];
    for (int a = 0; a < 9; a++) {
        hardEnemies[a].enemyX = (28*a);
        hardEnemies[a].enemyY = 160;
    }
    hardEnemies[3].enemyX = 240;
    hardEnemies[6].enemyX = 240;

    //player setup
    struct Players player;
    player.playerX = 120;
    player.playerY = 136;

    //fast enemy "boss" setup
    struct FastEnemy fast;
    fast.fastX = 0;
    fast.fastY = 30;

    // REG_DISPCNT = MODE3 | BG2_ENABLE;
    //inititalize title screen
    print_text_on_vga(10, 20, "GALAGA ");
    drawImage3(0, 0, 240, 160, titlescreen);
    while(1) {
        if (KEY_DOWN_NOW(BUTTON_START)) {

```

```

        break;
    }
}
//start black screen for drawing
for (int i = 0; i < 240; i++) {
    for (int j = 0; j < 160; j++) {
        setPixel(i, j, BLACK);
    }
}
while(1) {
    //go back to title screen if select button is pressed
    if (KEY_DOWN_NOW(BUTTON_SELECT)) {
        //initialize();
        juego();
    }
    //avisa al proceso control que finalice el juego si el usuario selecciona la
tecla x
    if (KEY_DOWN_NOW(BUTTON_FIN)) {
        send(pid_control,cantPuntaje);
    }

    //player shots
    if (KEY_DOWN_NOW(BUTTON_A)) {
        if (shoots[curr_shot] == 0) {
            shoots[curr_shot] = 136*240 + player.playerX+9; /* 24 width player
*/
            curr_shot++;
            if (curr_shot >= N_SHOOTS)
                curr_shot = 0;
        };
    }
    //player movement input
    if (KEY_DOWN_NOW(BUTTON_LEFT) && (player.playerX > 0)) {
        player.playerX -= playerspeed;
    }
    if (KEY_DOWN_NOW(BUTTON_RIGHT) && (player.playerX <= 216)) {
        player.playerX += playerspeed;
    }
    if (KEY_DOWN_NOW(BUTTON_UP) && (player.playerY > 25)) {
        player.playerY -= playerspeed;
    }
    if (KEY_DOWN_NOW(BUTTON_DOWN) && (player.playerY <= 136)) {
        player.playerY += playerspeed;
    }
    waitForVBlank();
    sleepms(50);
    //draw player
    drawImage3(player.playerX, player.playerY, 24, 24, playerImage);
    drawHollowRect(player.playerX - 1, player.playerY - 1, 26, 26, BLACK);
    drawHollowRect(player.playerX - 2, player.playerY - 2, 28, 28, BLACK);
    //draw easy enemies with downward movement
    for (int a = 0; a < 9; a++) {

```

```

        easyEnemies[a].enemyY += enemyspeed;
        drawImage3(easyEnemies[a].enemyX, easyEnemies[a].enemyY, 20, 20, enemy);
        if (collision(easyEnemies[a].enemyX, easyEnemies[a].enemyY, 20, 20,
player.playerX, player.playerY)) {
            //se produjo una colisión, pierde una vida y se avisa al proceso
2(vida_puntaje)

            drawRect(easyEnemies[a].enemyX, easyEnemies[a].enemyY, 20, 20,
BLACK);

            easyEnemies[a].enemyY = 0;
            send(pid_p2, 1);
        }
        if (easyEnemies[a].enemyY >= 160) {
            easyEnemies[a].enemyY = 0;
        }
    }

    //draw shots
    for (int i = 0; i < N_SHOOTS; i++) {
        if (shoots[i] != 0) {
            drawRect((shoots[i] % 240), (shoots[i] / 240)+4, 5, 5, BLACK);
            drawImage3((shoots[i] % 240), (shoots[i] / 240), 5, 5, shoot);
            shoots[i] = shoots[i]-240*4;
            if (shoots[i] <=0)    shoots[i]=0;
        }

        // check hits of shoots
        for (int j = 0; j < 9; j++) {
            if ( (shoots[i] != 0) && collision(easyEnemies[j].enemyX,
easyEnemies[j].enemyY, 15, 15, shoots[i] % 240, shoots[i] / 240)) {
                drawRect(easyEnemies[j].enemyX, easyEnemies[j].enemyY, 20,
20, BLACK);

                drawRect((shoots[i] % 240), (shoots[i] / 240)+4, 5, 5, BLACK);
                //drawImage3((shoots[i] % 240), (shoots[i] / 240), 5, 5,
BLACK);

                easyEnemies[j].enemyY = 0;
                shoots[i] = 0;
                //el jugador logro dispararle al enemigo, avisa al proceso
2(vida_puntaje) que suma puntos
                send(pid_p2, puntos);
            }
        }
    }

    //draw hard enemies
    for (int a = 0; a < 9; a++) {
        hardEnemies[a].enemyY += enemyspeed;
        drawImage3(hardEnemies[a].enemyX, hardEnemies[a].enemyY, 20, 20, enemy);
        if (collision(hardEnemies[a].enemyX, hardEnemies[a].enemyY, 20, 20,
player.playerX, player.playerY)) {
            //se produjo una colisión, pierde una vida y se avisa al proceso
2(vida_puntaje)

```

```

        drawRect(hardEnemies[a].enemyX, hardEnemies[a].enemyY, 20, 20,
BLACK);

        easyEnemies[a].enemyY = 0;
        send(pid_p2, 1);
    }
    if (hardEnemies[a].enemyY >= 228) {
        hardEnemies[a].enemyY = 0;
    }
    if ((hardEnemies[a].enemyY >= 200) && (easyEnemies[a].enemyY <=45)) {
        hardEnemies[a].enemyY = 160;
    }
    //space enemies apart
    if ((hardEnemies[a].enemyY >= 200) && (easyEnemies[a].enemyY <=45)) {
        hardEnemies[a].enemyY = 160;
    }
    if ((easyEnemies[a].enemyY >= 120) && (hardEnemies[a].enemyY >=170)) {
        hardEnemies[a].enemyY = 160;
    }
}
//draw fast enemy
drawImage3(fast.fastX, fast.fastY, 15, 15, boss);
drawHollowRect(fast.fastX - 1, fast.fastY - 1, 17, 17, BLACK);
drawHollowRect(fast.fastX - 2, fast.fastY - 2, 19, 19, BLACK);
if(collision(fast.fastX, fast.fastY, 15, 15, player.playerX, player.playerY)) {
    //se produjo una colisión, pierde una vida y se avisa al proceso
2(vida_puntaje)

    drawRect(fast.fastX, fast.fastY, 20, 20, BLACK);
    fast.fastX = 0;
    //fast.fastY = player.playerY - 20;
    send(pid_p2, 1);
}
//comentado por rafa
fast.fastX += fastXSpeed;
fast.fastY += fastYSpeed;
if (fast.fastX >= 240) {
    fast.fastX = 0;
}
if (fast.fastY >= 200) {
    fast.fastY = player.playerY - 20;
}
}
return 0;
}

```

```

int collision(u16 enemyX, u16 enemyY, u16 enemyWidth, u16 enemyHeight, u16 playerX, u16
playerY) {
    //check if bottom right corner of enemy is within player
    if (((enemyX + enemyWidth) > playerX) && ( (enemyY + enemyHeight)
> playerY ) && ((enemyX + enemyWidth) < (playerX + 24))

```

```

        && ((enemyY + enemyWidth) < (playerY + 24))) {
            return 1;
        }
        //check bottom left corner of enemy
        if ( (enemyX < (playerX + 24))
            && (enemyX > playerX)
            && ((enemyY + enemyHeight) > playerY)
            && ((enemyY + enemyHeight) < (playerY + 24))
            ) {
            return 1;
        }
        //check top left corner
        if ( (enemyX < (playerX + 24))
            && (enemyX > playerX)
            && (enemyY > playerY)
            && (enemyY < (playerY + 24))
            ) {
            return 1;
        }
        //check top right corner
        if ( ((enemyX + enemyWidth) < (playerX + 24))
            && ((enemyX + enemyWidth) > playerX)
            && (enemyY > playerY)
            && (enemyY < (playerY + 24))
            ) {
            return 1;
        }
        return 0;
    }

//jugador finaliza el juego porque pierde o porque apreta la tecla x
void endGame() {
    //start Game Over State
    drawImage3(0, 0, 240, 160, gameover);
    drawHollowRect(0, 0, 240, 160, WHITE);
    while(1) {
        if (KEY_DOWN_NOW(BUTTON_SELECT)) {
            galaga();
            break;
        }
        if (KEY_DOWN_NOW(BUTTON_START)) {
            galaga();
            break;
        }
    }
}

//CREACIÓN DE LOS PROCESOS

int galaga(void){

```



```

pid_control = create(controlJuego,1024,20,"proceso control",0);
resume(pid_control);
sleep(2);
}

```

parte agregada al for de shoots:

```

for (int j = 0; j < 9; j++) {
    if(shoots[i] != 0){
        if(collision(easyEnemies[j].enemyX, easyEnemies[j].enemyY, 15,
15, shoots[i] % 240, shoots[i] / 240)){
            drawRect(easyEnemies[j].enemyX, easyEnemies[j].enemyY,
20, 20, BLACK);
            drawRect((shoots[i] % 240), (shoots[i] / 240)+4, 5, 5,
BLACK);
            //drawImage3((shoots[i] % 240), (shoots[i] / 240), 5, 5,
BLACK);
            easyEnemies[j].enemyY = 0;
            shoots[i] = 0;
            //el jugador logro dispararle al enemigo, avisa al
proceso 2 que sume puntos
            send(pid_p2, puntos);
        }else{
            if(collision(hardEnemies[j].enemyX,
hardEnemies[j].enemyY, 20, 20, shoots[i] % 240, shoots[i] / 240)){
                drawRect(hardEnemies[j].enemyX,
hardEnemies[j].enemyY, 20, 20, BLACK);
                drawRect((shoots[i] % 240), (shoots[i] / 240)+4,
5, 5, BLACK);
                hardEnemies[j].enemyY = 0;
                shoots[i] = 0;
                //el jugador logro dispararle al enemigo, avisa
al proceso 2 que sume puntos
                send(pid_p2, puntos);
            }
        }
    }
}
}
}

```

# Trabajo Práctico N°4

**Ejercicio 1.** Programa para convertir una imagen a escala de grises, en PC UNIX/Linux.

- a. ¿Qué significa el acrónimo gcc?. ¿Desde cuándo está el proyecto?. ¿Cuál es su objetivo?

El acrónimo "GCC" en Unix/Linux se refiere al **"GNU Compiler Collection"**, que es un conjunto de compiladores de código abierto desarrollados por el Proyecto GNU. GCC es ampliamente utilizado para compilar programas escritos en lenguajes como C, C++, Objective-C, Fortran, Ada y otros.

El Proyecto GNU fue anunciado por Richard Stallman el **27 de septiembre de 1983**. Desde entonces, ha estado desarrollando software libre y promoviendo los derechos de los usuarios de software. El objetivo principal del Proyecto GNU es crear un sistema operativo completo y totalmente libre, compatible con Unix, conocido como GNU. Este sistema operativo GNU se utiliza ampliamente junto con el núcleo Linux en muchas distribuciones de software libre y de código abierto, como GNU/Linux.

- b. ¿Qué es POSIX?

POSIX (Portable Operating System Interface) es un **conjunto de estándares** que define la interfaz de programación de aplicaciones (API), la interfaz de línea de comandos (CLI) y los servicios del sistema operativo para sistemas operativos tipo Unix. Estos estándares fueron desarrollados con el objetivo de asegurar la portabilidad del software entre diferentes sistemas operativos Unix y compatibles con Unix. POSIX especifica las interfaces estándar para funciones como la gestión de archivos, la concurrencia, la comunicación entre procesos, la gestión de memoria y la entrada/salida. Esto permite a los desarrolladores escribir aplicaciones que puedan ejecutarse en cualquier sistema compatible con POSIX sin necesidad de modificar el código fuente.

La adherencia a los estándares POSIX es común en sistemas operativos como Unix, Linux y sistemas basados en BSD, lo que facilita la interoperabilidad entre ellos y la portabilidad de las aplicaciones desarrolladas para estos sistemas.

- c. Utilice el siguiente programa en lenguaje C:

Observe la imagen input.bmp que se encuentra en la carpeta convertir/. Compilar y ejecutar el programa convertir\_a\_gris.c en un sistema UNIX/Linux.

- d. ¿Considera que este programa podría ser portado fácilmente a MAC OS? ¿O a FreeBSD?. Si/No, porqué. Justifique su respuesta.

Muchos programas diseñados para Unix/Linux pueden portarse a macOS con relativamente pocas modificaciones, especialmente si siguen las mejores prácticas de programación y se adhieren a estándares ampliamente aceptados como POSIX. macOS, al igual que Linux, se basa en una variante de Unix (en particular, FreeBSD), por lo que muchos programas de Unix/Linux pueden funcionar sin problemas en macOS.

Sin embargo, puede haber diferencias en las herramientas y bibliotecas disponibles entre los sistemas operativos, lo que puede requerir ajustes en el código o la adaptación de ciertas partes del programa. Además, algunos programas pueden depender de características específicas de Linux que no estén presentes en macOS, lo que requeriría una reescritura más sustancial.

En resumen, si bien muchos programas de Unix/Linux pueden portarse a macOS, la facilidad de la portabilidad dependerá de la complejidad y las dependencias del programa en cuestión

Es posible porque ambos están basados en UNIX, y comparten muchas similitudes de su estructura y biblioteca. Pero hay que tener en consideración que las bibliotecas que utilicemos estén en MACOS o en FREEBSD.

2)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
```

```

#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>

#define BMP_FILE "input.bmp"
#define GRAYSCALE_FILE "output_grayscale.bmp"

#pragma pack(push, 1)
typedef struct {
    unsigned char magic[2];
    unsigned int size;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offset;
} BMPHeader;

typedef struct {
    unsigned int headerSize;
    int width;
    int height;
    unsigned short planes;
    unsigned short bpp;
    unsigned int compression;
    unsigned int imageSize;
    int xPixelsPerM;
    int yPixelsPerM;
    unsigned int colorsUsed;
    unsigned int colorsImportant;
} BMPInfoHeader;
#pragma pack(pop)

unsigned char nueva_imagen [2000*2000*3];

void convertir(int in_fd, int out_fd, BMPHeader h, BMPInfoHeader infoh, int inicio, int
fin, int num)
{
    int width = infoh.width;
    int height = infoh.height;
    unsigned char gray;
    int bandera = 1;
    int primero;
    int padding = (4 - (width * 3) % 4) % 4; // Calculating padding size
    unsigned char pixel[3];
    int tamCabecera = (int)(sizeof(BMPHeader) + sizeof(BMPInfoHeader));

    lseek(in_fd, width * height * num, SEEK_SET);

    for (int y = inicio; y < fin; y++) {
        for (int x = 0; x < width; x++) {
            read(in_fd, pixel, 3);

```

```

        gray = (unsigned char)(0.3 * pixel[2] + 0.59 * pixel[1] + 0.11 *
pixel[0]);
        if(bandera){
            primero = y*(width*3+padding);
            bandera = 0;
        }
        nueva_imagen[y*(width*3+padding) + x*3]= gray;
        nueva_imagen[y*(width*3+padding) + x*3 + 1] = gray;
        nueva_imagen[y*(width*3+padding) + x*3 + 2] = gray;
    }
    lseek(in_fd, padding, SEEK_CUR); // Skipping padding
}
lseek(out_fd, (width * height * num) + tamCabecera, SEEK_SET);
write(out_fd, &nueva_imagen[primero], height * width);
}

void something_wrong(int fd, const char *m){
    if (fd > 0)
        close(fd);
    printf("%s\n", m),
    exit(1);
}

void long_running_function() {
    for (long i = 0; i < 1000000000; i++); // Simulación de una operación prolongada
}

int main() {
    int pid_hijo1, pid_hijo2, pid_hijo3, status;
    clock_t start, end;
    double cpu_time_used;
    // Iniciar el reloj
    start = clock();

    // Llamar a la función cuyo tiempo de ejecución queremos medir
    long_running_function();
    int in_fd = open(BMP_FILE, O_RDONLY);
    if (in_fd < 0) {
        something_wrong(in_fd, "Error open");
    }

    BMPHeader h;
    read(in_fd, &h, sizeof(BMPHeader));

    if (h.magic[0] != 'B' || h.magic[1] != 'M') {
        something_wrong(in_fd, "No es BMP");
    }

    BMPInfoHeader infoh;
    read(in_fd, &infoh, sizeof(BMPInfoHeader));

    if (infoh.bpp != 24 || infoh.compression != 0) {

```

```

    something_wrong(in_fd, "No 24-bit BMP");
}

int out_fd = open(GRAYSCALE_FILE, O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (out_fd < 0) {
    something_wrong(in_fd, "error en output");
}

write(out_fd, &h, sizeof(BMPHeader));
write(out_fd, &infoh, sizeof(BMPInfoHeader));

pid_hijo1 = fork();
if(pid_hijo1 == 0){
    // Debe de abrir cada in y out ya que los hijos no lo poseen
    int in_fd1 = open(BMP_FILE, O_RDONLY);
    int out_fd1 = open(GRAYSCALE_FILE, O_WRONLY, 0644);
    convertir(in_fd1, out_fd1, h, infoh, 0, 354, 0);
    if (out_fd1 < 0) {
        something_wrong(in_fd1, "error en output");
    }
    close(in_fd1);
    close(out_fd1);
}else{ // Padre
    pid_hijo2 = fork();
    if(pid_hijo2 == 0){
        // Debe de abrir cada in y out ya que los hijos no lo poseen
        int in_fd2 = open(BMP_FILE, O_RDONLY);
        int out_fd2 = open(GRAYSCALE_FILE, O_WRONLY, 0644);
        convertir(in_fd2, out_fd2, h, infoh, 354, 709, 1);
        if (out_fd2 < 0) {
            something_wrong(in_fd2, "error en output");
        }
        close(in_fd2);
        close(out_fd2);
    }else{ // Padre
        pid_hijo3 = fork();
        if(pid_hijo3 == 0){
            // Debe de abrir cada in y out ya que los hijos no lo poseen
            int in_fd3 = open(BMP_FILE, O_RDONLY);
            int out_fd3 = open(GRAYSCALE_FILE, O_WRONLY, 0644);
            convertir(in_fd3, out_fd3, h, infoh, 710, 1063, 2);
            if (out_fd3 < 0) {
                something_wrong(in_fd3, "error en output");
            }
            close(in_fd3);
            close(out_fd3);
        }else{
            waitpid(pid_hijo1, NULL, 0);
            waitpid(pid_hijo2, NULL, 0);
            waitpid(pid_hijo3, NULL, 0);
        }
    }
}

```

```

        close(in_fd);
        close(out_fd);

        kill(pid_hijo1, SIGTERM);
        kill(pid_hijo2, SIGTERM);
        kill(pid_hijo3, SIGTERM);

        printf("Imagen en gris generada. %s\n", GRAYSCALE_FILE);
        // Detener el reloj
        end = clock();

        // Calcular el tiempo utilizado
        cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;

        printf("El tiempo de ejecución es %f segundos\n",
cpu_time_used);

        exit(EXIT_SUCCESS);
    }

}

}

}

```

c. En el programa, ¿Cuáles llamadas a funciones son funciones de la biblioteca de C que son simplemente útiles de manera general y cuáles son funciones de la biblioteca de C que realizan llamadas al sistema?

Las funciones que hacen llamadas al sistema suelen estar relacionadas con operaciones de entrada y salida, gestión de archivos, gestión de memoria, administración de procesos, y más. Algunas de las funciones de la biblioteca estándar de C que hacen llamadas al sistema incluyen: (las del programa en color azul)

#### 1. Funciones de entrada/salida (`stdio.h`):

- `printf()`, `scanf()`, `fopen()`, `fclose()`, `fread()`, `fwrite()`, etc.

#### 2. Funciones de gestión de archivos (`stdio.h`, `fcntl.h`, `unistd.h`):

- `open()`, `close()`, `read()`, `write()`, `lseek()`, etc.

#### 3. Funciones de gestión de directorios (`dirent.h`, `unistd.h`):

- `opendir()`, `readdir()`, `closedir()`, etc.

#### 4. Funciones de gestión de procesos (`unistd.h`, `sys/types.h`, `sys/wait.h`):

- `fork()`, `exec()`, `wait()`, `exit()`, `kill()` etc.

#### 5. Funciones de gestión de memoria (`stdlib.h`, `unistd.h`, `sys/mman.h`):

- `malloc()`, `free()`, `mmap()`, `munmap()`, etc.

#### 6. Funciones de gestión de tiempo (`time.h`, `sys/time.h`):

- `time()`, `gettimeofday()`, `sleep()`, etc.

## 7. Funciones de red (`sys/socket.h`, `netinet/in.h`, `arpa/inet.h`):

- `socket()`, `bind()`, `connect()`, `listen()`, `accept()`, etc.

3b)

PAU Y TONY

```
// Ejercicio 3 b
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>
#include <time.h>

#define BMP_FILE "input.bmp"
#define GRAYSCALE_FILE "output_grayscale.bmp"

#pragma pack(push, 1)
typedef struct {
    unsigned char magic[2];
    unsigned int size;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offset;
} BMPHeader;

typedef struct {
    unsigned int headerSize;
    int width;
    int height;
    unsigned short planes;
    unsigned short bpp;
    unsigned int compression;
    unsigned int imageSize;
    int xPixelsPerM;
    int yPixelsPerM;
    unsigned int colorsUsed;
    unsigned int colorsImportant;
} BMPInfoHeader;
#pragma pack(pop)

typedef struct {
    BMPInfoHeader infoh;
    BMPHeader h;
    int in_fd;
    int out_fd;
```

```

    int inicio;
    int fin;
    int num;
} thread_args;

unsigned char nueva_imagen [2000*2000*3];

void long_running_function() {
    for (long i = 0; i < 1000000000; i++); // Simulación de una operación prolongada
}

void *convertir(void *argumentos) {

    thread_args *args = (thread_args*)argumentos;

    int width = args->infoh.width;
    int height = args->infoh.height;
    unsigned char gray;
    int bandera = 1;
    int primero;
    int padding = (4 - (width * 3) % 4) % 4; // Calculating padding size
    unsigned char pixel[3];
    int tamCabecera = (int)(sizeof(BMPHeader) + sizeof(BMPInfoHeader));

    lseek(args->in_fd, width * height * args->num, SEEK_SET);

    for (int y = args->inicio; y < args->fin; y++) {
        for (int x = 0; x < width; x++) {
            read(args->in_fd, pixel, 3);
            gray = (unsigned char)(0.3 * pixel[2] + 0.59 * pixel[1] + 0.11 *
pixel[0]);
            if(bandera){
                primero = y*(width*3+padding);
                bandera = 0;
            }
            nueva_imagen[y*(width*3+padding) + x*3]= gray;
            nueva_imagen[y*(width*3+padding) + x*3 + 1] = gray;
            nueva_imagen[y*(width*3+padding) + x*3 + 2] = gray;
        }
        lseek(args->in_fd, padding, SEEK_CUR); // Skipping padding
    }
    lseek(args->out_fd, (width * height * args->num) + tamCabecera, SEEK_SET);
    write(args->out_fd, &nueva_imagen[primero], height * width);
}

void something_wrong(int fd, const char *m)
{
    if (fd > 0)
        close(fd);
    printf("%s\n", m),
    exit(1);
}

```



```

int main()
{
    clock_t start, end;
    double cpu_time_used;
    // Iniciar el reloj
    start = clock();

    // Llamar a la función cuyo tiempo de ejecución queremos medir
    long_running_function();
    int in_fd = open(BMP_FILE, O_RDONLY);
    if (in_fd < 0) {
        something_wrong(in_fd, "Error open");
    }

    BMPHeader h;
    read(in_fd, &h, sizeof(BMPHeader));

    if (h.magic[0] != 'B' || h.magic[1] != 'M') {
        something_wrong(in_fd, "No es BMP");
    }

    BMPInfoHeader infoh;
    read(in_fd, &infoh, sizeof(BMPInfoHeader));

    if (infoh.bpp != 24 || infoh.compression != 0) {
        something_wrong(in_fd, "No 24-bit BMP");
    }

    int out_fd = open(GRAYSCALE_FILE, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (out_fd < 0) {
        something_wrong(in_fd, "error en output");
    }

    // Escribimos la cabecera
    write(out_fd, &h, sizeof(BMPHeader));
    write(out_fd, &infoh, sizeof(BMPInfoHeader));

    // Creo un arreglo de los hilos
    pthread_t t_hilos[3];
    // Creo un arreglo de los argumentos para cada hilo
    thread_args args[3];

    // Recorro cada posición del arreglo para setear los argumentos para cada hilo
    for(int i = 0; i < 3; i++){
        args[i].infoh = infoh;
        args[i].h = h;
        // Debe de abrir cada in y out ya que los hilos no lo poseen
        args[i].in_fd = open(BMP_FILE, O_RDONLY);
        if (args[i].in_fd < 0) {
            something_wrong(args[i].in_fd, "Error open");
        }
    }
}

```

```

        args[i].out_fd = open(GRAYSCALE_FILE, O_WRONLY, 0644);
        if (args[i].out_fd < 0) {
            something_wrong(args[i].out_fd, "error en output");
        }
        args[i].num = i;
        args[i].inicio = 354 * i;
        args[i].fin = 354 * (i+1);
    }

    // Inicializo cada hilo
    for (int i = 0; i < 3; i++){
        pthread_create(&t_hilos[i], NULL, convertir, (void*)&args[i]);
    }

    // Espero a que cada hilo termine
    for (int i = 0; i < 3; i++){
        pthread_join(t_hilos[i], NULL);
    }

    close(in_fd);
    close(out_fd);

    // Detener el reloj
    end = clock();
    // Calcular el tiempo utilizado
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    printf("El tiempo de ejecución es %f segundos\n", cpu_time_used);

    printf("Imagen en gris generada. %s\n", GRAYSCALE_FILE);
    exit(EXIT_SUCCESS);
}

```

OTRA FORMA (alba) es mucho mejor!!!

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>

#define BMP_FILE "input.bmp"
#define GRAYSCALE_FILE "output_grayscale.bmp"

#pragma pack(push, 1)
typedef struct {
    unsigned char magic[2];
    unsigned int size;

```

```

    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offset;
} BMPHeader;

typedef struct {
    unsigned int headerSize;
    int width;
    int height;
    unsigned short planes;
    unsigned short bpp;
    unsigned int compression;
    unsigned int imageSize;
    int xPixelsPerM;
    int yPixelsPerM;
    unsigned int colorsUsed;
    unsigned int colorsImportant;
} BMPInfoHeader;
#pragma pack(pop)

//estructura para almacenar los parámetros de la función convertir
typedef struct {
    int num; //se usará para que cada hilo sepa desde qué fila deberá procesar
    BMPHeader h;
    BMPInfoHeader infoh;
} ThreadArgs;

unsigned char nueva_imagen [2000*2000*3];
int pos_arreglo; //indicará cuanto se cargó de la imagen

void *convertir(void * arg)
{
    //convertimos el argumento al tipo deseado
    ThreadArgs argum = *((ThreadArgs *) arg);

    //recuperamos los parametros
    int num = argum.num;
    BMPHeader h = argum.h;
    BMPInfoHeader infoh = argum.infoh;

    int width = infoh.width;

    int inicio = (pos_arreglo/3) * num; //desde qué posicion del arreglo leera el hilo
    int fin = (pos_arreglo/3) + inicio; //hasta qué posicion del arreglo leera el hilo

    //printf("pixeles inicio: %i",inicio);
    //printf("pixeles fin: %i",fin);

    int padding = (4 - (width * 3) % 4) % 4; // Calculating padding size
    unsigned char pixel[3];

```

```

    for (int y = inicio; y < fin; y +=3) {
        //tomamos los colores del pixel (sus bytes corresponden a las 3 posiciones
consecutivas)
        unsigned char blue = nueva_imagen[y];
        unsigned char green = nueva_imagen[y+1];
        unsigned char red = nueva_imagen[y+2];

        //aplicamos la escala a gris
        unsigned char gray = (unsigned char)(0.3 * red + 0.59 * green + 0.11 * blue);

        //sobreescribimos en e el arreglo
        nueva_imagen[y] = gray;
        nueva_imagen[y+1] = gray;
        nueva_imagen[y+2] = gray;
    }
}

void copiar_imagen(int in_fd, int out_fd, BMPHeader h, BMPInfoHeader infoh){
    int width = infoh.width;
    int height = infoh.height;
    int padding = (4 - (width * 3) % 4) % 4; // Calculating padding size
    unsigned char pixel[3];
    int y,x;

    for (y = 0; y < height; y++) {
        for ( x = 0; x < width; x++) {
            read(in_fd, pixel, 3);

            nueva_imagen[y*(width*3+padding) + x*3]= pixel[0];
            nueva_imagen[y*(width*3+padding) + x*3 + 1] = pixel[1];
            nueva_imagen[y*(width*3+padding) + x*3 + 2] = pixel[2];

            if((x == (width-1)) && (y == (height -1))){
                pos_arreglo = (y*(width*3+padding) + x*3+2); //almacenamos cuántos
pixeles de la imagen se procesaron
                //printf("la ultima posicion del arreglo con bytes cargados es:
%i",pos_arreglo);
            }
        }
        lseek(in_fd, padding, SEEK_CUR); // Skipping padding
    }
}

void something_wrong(int fd, const char *m)
{
    if (fd > 0)
        close(fd);
    printf("%s\n", m),
    exit(1);
}

```

```

int main()
{
    pthread_t tid[3]; //arreglo de hilos

    int in_fd = open(BMP_FILE, O_RDONLY);
    if (in_fd < 0) {
        something_wrong(in_fd, "Error open");
    }

    BMPHeader h;
    read(in_fd, &h, sizeof(BMPHeader));

    if (h.magic[0] != 'B' || h.magic[1] != 'M') {
        something_wrong(in_fd, "No es BMP");
    }

    BMPInfoHeader infoh;
    read(in_fd, &infoh, sizeof(BMPInfoHeader));

    if (infoh.bpp != 24 || infoh.compression != 0) {
        something_wrong(in_fd, "No 24-bit BMP");
    }

    int out_fd = open(GRAYSCALE_FILE, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (out_fd < 0) {
        something_wrong(in_fd, "error en output");
    }

    //creamos las estructuras con los parametros necesarios de cada hilo
    //del hilo1
    ThreadArgs args1;
    args1.num = 0;
    args1.h = h;
    args1.infoh = infoh;
    //del hilo2
    ThreadArgs args2;
    args2.num = 1;
    args2.h = h;
    args2.infoh = infoh;
    //del hilo3
    ThreadArgs args3;
    args3.num = 2;
    args3.h = h;
    args3.infoh = infoh;

    //el padre copia los bytes de la imagen a color en un arreglo compartido
    copiar_imagen(in_fd, out_fd, h, infoh);

    /* Create independent threads each of which will execute function */
    //cada hilo convertira a gris una porcion del arreglo

```

```

pthread_create( &tid[0], NULL, convertir, (void *) &args1);
pthread_create( &tid[1], NULL, convertir, (void *) &args2);
pthread_create( &tid[2], NULL, convertir, (void *) &args3);

/* Wait till threads are complete before main continues. */
for (int i = 0; i <= 2; i++)
    pthread_join(tid[i], NULL);

write(out_fd, &h, sizeof(BMPHeader));
write(out_fd, &infoh, sizeof(BMPInfoHeader));
write(out_fd, &nueva_imagen[0], infoh.width * infoh.height * 3);

close(in_fd);
close(out_fd);

printf("Imagen en gris generada. %s\n", GRAYSCALE_FILE);
exit(0);
}

```

#### Ejercicio 4. Responda

- Utilice el comando time para medir los tiempos de los 3 programas que convierten\_a\_gris (Ejercicio 1, 2 y 3).

#### EJERCICIO 2 ORIGINAL

```

pcoronel@debian:~/Descargas/convertir$ gcc -o ejecu original.c
pcoronel@debian:~/Descargas/convertir$ ./ejecu
Imagen en gris generada. output_grayscale.bmp
El tiempo de ejecución es 5.395710 segundos
pcoronel@debian:~/Descargas/convertir$ time ./ejecu
Imagen en gris generada. output_grayscale.bmp
El tiempo de ejecución es 5.322719 segundos

real    0m5.331s
user    0m2.737s
sys     0m2.589s

```

#### EJERCICIO 2 MODIFICADO

```

pcoronel@debian:~/Descargas/convertir$ gcc -o ejecu convertir_a_gris.
pcoronel@debian:~/Descargas/convertir$ ./ejecu
Imagen en gris generada. output_grayscale.bmp
El tiempo de ejecución es 2.351524 segundos
pcoronel@debian:~/Descargas/convertir$ time ./ejecu
Imagen en gris generada. output_grayscale.bmp
El tiempo de ejecución es 2.364423 segundos

real    0m4,230s
user    0m2,665s
sys     0m2,898s

```

### EJERCICIO 3

```

pcoronel@debian:~/Descargas/convertir$ gcc -o ejecu ej3_convertir.c -lpthread
pcoronel@debian:~/Descargas/convertir$ ./ejecu
El tiempo de ejecución es 5.492096 segundos
Imagen en gris generada. output_grayscale.bmp
pcoronel@debian:~/Descargas/convertir$ time ./ejecu
El tiempo de ejecución es 5.548730 segundos
Imagen en gris generada. output_grayscale.bmp

real    0m5,576s
user    0m2,922s
sys     0m2,631s

```

- b. ¿Cuál versión fue la que tomó menos tiempo?. ¿Existe gran diferencia de ganancia entre hacerlo con procesos que con threads?. ¿Para qué tipo de programas puede que la ganancia sea mejor con threads?. Analice cuál de los mecanismos (un único proceso, varios procesos o varios threads) debería ser el más eficiente en cuanto a rendimiento (menor tiempo de ejecución).
- c. ¿Por qué no existe en el sistema operativo XINU una llamada al sistema específica para crear threads o hilos?.

En el sistema operativo Xinu, no existe una llamada al sistema específica para crear threads porque Xinu fue diseñado originalmente como un sistema operativo educativo y de investigación que se enfoca en la simplicidad y la comprensión de los principios fundamentales de los sistemas operativos. En lugar de tener una llamada al sistema dedicada para la creación de threads, Xinu sigue un enfoque más básico donde el manejo de concurrencia se basa principalmente en el uso de procesos ligeros (o "lwildows" en Xinu) en lugar de threads completos. Estos procesos ligeros comparten el mismo espacio de dirección que el proceso padre y se ejecutan en el mismo contexto de procesador. El cambio de contexto entre los procesos ligeros se realiza mediante una función de planificación en el núcleo del sistema operativo.

## Trabajo Práctico N°5

```

#include <xinu.h>
#include <keyboard.h>
sid32 sem;
int buffer_cantidad;
char buffer_tecle[10]; //inicializo el buffer de manera global
char tecla_actual;
int buffer_frente;
int buffer_fin;
sid32 mutex;
int pid;
int vacia;
void kbd_init() {

    /*
    Ejercicio tp5

    buffer_cantidad = 10;
    sem = semcreate(0);
    mutex_init();
    vacia = 1;
    buffer_frente = 0;
    buffer_fin = 0;
    */

    mutex_init(); //Para parcial 2023

}
void mutex_init(){
    mutex = semcreate(1);
}

void mutex_lock(){
    wait(mutex);
    pid = getpid();
}

void mutex_unlock(){
    if(getpid() == pid){
        signal(mutex);
    }
}

```

#### ***xinu-pc-galaga/device/kbd/kbdhandler.c***

```

#include <xinu.h>
#include <keyboard.h>
extern char buffer_tecle[];
extern int buffer_fin;
extern int buffer_frente;
extern int buffer_cantidad;
extern sid32 sem;
extern int vacia;

```



```

extern int pid;
void kbdhandler() {
    char t[80];

    unsigned char scancode;
    unsigned int shift_key = 0;

    int i = 10;
    char buf[80];
    char buf1[80];

    scancode = get_scancode();

    send(pid, scancode); //Para parcial 2023

    /*    Para TP5
    if((vacia == 1) || (buffer_frente != buffer_fin)){

        buffer_tecla[buffer_fin] = scancode;
        buffer_fin = (buffer_fin + 1) % buffer_cantidad;
        vacia = 0;

        sprintf(t, " tecla actual kbd: 0x%x", scancode);
        print_text_on_vga(10, 300, t);

        sprintf(buf, "buffer_frente: %d", buffer_frente);
        print_text_on_vga(10, 400, buf);

        sprintf(buf, "buffer_fin: %d", buffer_fin);
        print_text_on_vga(10, 420, buf);

        signal(sem);

        sprintf(buf, "SEM_COUNT : %d", semcount(sem));
        print_text_on_vga(10, 500, buf);

    } else{
        sprintf(t, " tecla actual kbd: 0x%x", scancode);
        print_text_on_vga(10, 300, t);
        sprintf(buf1, "BUFFER LLEN0, SEM_COUNT : %d", semcount(sem));
        print_text_on_vga(10, 500, buf1);
    }
    */

    sprintf(t, " tecla actual kbd: 0x%x", scancode); //Para parcial 2023
    print_text_on_vga(10, 300, t); //Para parcial 2023
}

```

### *xinu-pc-galaga/device/kbd/kbdgetc.c*

```
char kbd_getc(struct dentry *devptr) { //esta función hay que crearla
    unsigned char tecla;
    /*
    ejercicio tp5

    if(getpid() == pid){
        wait(sem);
        tecla = buffer_tecla[buffer_frente];
        buffer_frente = (buffer_frente + 1) % buffer_cantidad;
        if(buffer_frente == buffer_fin){
            vacia = 1;
        }
    }
    */
    tecla = receive(); // Por comunicacion de procesos (Parcial 2023)

    return (devcall)tecla;
}
```

### *xinu-pc-galaga/device/kbd/kbdread.c*

```
devcall kbdread (
    struct dentry *devptr, /* Entry in device switch table */
    char *buffer, /* Address of buffer */
    uint32 count /* Length of buffer */
)
{
    char t[80];
    if(count <= 0){
        return SYSERR;
    }
    *buffer = kbdgetc(devptr);

    if(*buffer != NULL){
        sprintf(t, "se consumo la letra: 0x%x", *buffer);
        print_text_on_vga(10, 320, t);
    }

    return OK;
}
```

### *xinu-pc-galaga/device/kbd/kbdopen.c*

```
devcall kbdopen (
    struct dentry *devptr, /* Entry in device switch table */
    char *name, /* Unused for a kbd */
    char *mode /* Unused for a kbd */
)
{
    /* No action -- just return the device descriptor */
}
```

```
mutex_lock();  
}
```

*xinu-pc-galaga/device/kbd/kbdclose.c*

```
devcall    kbdclose (  
    struct dentry    *devptr    /* Entry in device switch table    */  
    )  
{  
    mutex_unlock();  
}
```

## Segundo Parcial 2023

### Ejercicio 3

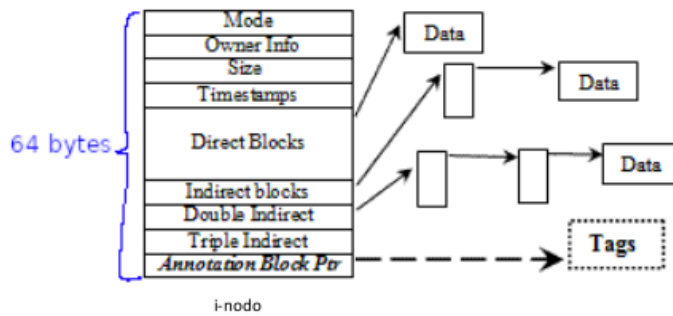
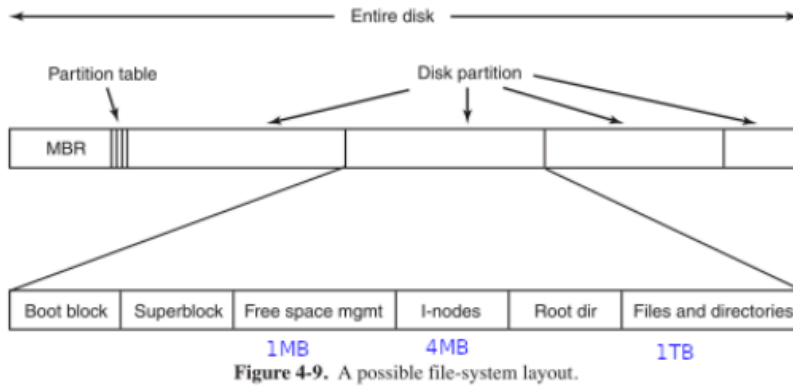
Responda:

1. Si las páginas del sistema de memoria virtual en los sistemas Linux de los laboratorios son de 4KB, ¿cuales son los 12 bits menos significativos de la dirección física que corresponde a la dirección virtual obtenida en el ejercicio 1.b. ? Es decir, luego de que el sistema realizó la traducción de la dirección virtual a física. Utilice el sistema hexadecimal para especificar la respuesta.
2. En C en Linux existen las funciones `shm_open()`, `ftruncate()` y `mmap()` para crear un segmento de memoria compartida que luego se puede compartir con otros procesos para comunicación entre procesos.

Indique en XINU cuáles serían las funciones en C equivalentes, y si no existen, explique cómo se logra ese mecanismo de comunicación entre procesos.

En XINU no existen directamente las funciones equivalentes a `shm_open()`, `ftruncate()`, y `mmap()` que se utilizan en Linux para la creación y manejo de memoria compartida entre procesos. Sin embargo, se pueden emplear diferentes mecanismos para lograr comunicación entre procesos. En XINU, la comunicación entre procesos generalmente se logra mediante colas de mensajes, semáforos, o la creación de espacios de memoria compartida utilizando las funciones del sistema.

3. Sean los siguientes diagramas de un UNIX file system:



¿Cuántos archivos máximo puede almacenar este sistema de archivos?

## Recu Segundo Parcial 2023

### Ejercicio 1:

a.

Implemente en Linux dos programas:

1. Un programa crea **memoria compartida**, y carga en la memoria compartida la foto cat.pgm, usando funciones para acceder a archivos en el **sistema de archivos**.
2. Un segundo programa lee de la memoria compartida la foto, y la guarda en un nuevo archivo de manera invertida.
  - Para invertir la imagen se debe mantener la cabecera intacta, y luego deberán estar los bytes de los píxeles invertidos (el último byte de pixel de cat.pgm será el primer byte de pixel en cat2.pgm, etc).

El formato pgm de la imagen de ejemplo es sencillo:

- Los primeros 15 bytes son la cabecera del formato del archivo de imagen.
- Los siguientes bytes son los valores de cada pixel de la imagen.

**Utilice memoria compartida entre los dos programas.**

**El tamaño de la memoria compartida es estático y conocido: 50261 bytes (que es el tamaño del archivo cat.pgm)**

Las funciones de C para crear un archivo nuevo y escribir 5 bytes son (suponga que imagen es un ARREGLO).

```
#include <unistd.h>
```

```
const char *nuevo = "cat2.pgm";
fd = open(nuevo, O_RDWR | O_CREAT | O_TRUNC, 0666); /* le pide al sistema operativo crear un archivo nuevo */
```

`write(fd, &imagen[6], 5); /* escribe en el archivo con descriptor fd 5 bytes, a partir del septimo byte del arreglo imagen[ ] */`

b. Agregue al SEGUNDO programa sentencias para que emita por pantalla la dirección virtual del inicio del segmento de memoria compartida. ¿Cuál es esa dirección? (recuerde que son direcciones de 64 bits).

c.

1. Utilizando `/proc/PID/maps` del segundo programa en ejecución indique las direcciones virtuales del segmento donde se encuentra la memoria compartida (como tip podría observar maps antes de obtener acceso a la memoria compartida, y luego de haber tenido acceso, y detectar cuál es o son los segmentos nuevos).
2. Indique también a qué segmento de memoria del proceso pertenece.

**Según facu, hay que ver el `getpid()` del proceso y poner `/proc/nropid/maps` en otra consola para que funcione mientras se esta ejecutando el otro proceso**

```
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h> /* Para ftruncate */
#include <fcntl.h> /* Para constantes O_* */

#define PGM_FILE "cat.pgm"

//PROGRAMA 1

int main(){
    const char *name = "memoria";
    const int SIZE = 50261;
    void *ptr;

    //creamos el segmento de memoria compartida
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    //configuramos el tamaño de la memoria compartida
    ftruncate(shm_fd,SIZE);

    //now map the shared memory segment in the address space of the process
    ptr = mmap(0,SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,0);

    if(ptr == MAP_FAILED){
        printf("Map FAILED");
        return -1;
    }

    //abrimos la imagen

    int in_fd = open(PGM_FILE,O_RDONLY);

    if (in_fd < 0) {
        printf("Error open");
        return -1;
    }

    //copiamos la imagen en el segmento de memoria compartida
    read(in_fd,ptr,SIZE);

    if (close(in_fd) < 0){
        printf("Error al cerrar el archivo");
    }

    return 0;
}
```

## Programa 2

```
#include <stdio.h>
#include <sys/mman.h>
#include <stdlib.h> /* malloc */
#include <unistd.h> /* Para ftruncate */
#include <fcntl.h> /* Para constantes O_* */
#include <string.h> /* para el memcpy*/

#define NEW_FILE "cat2.pgm"

//PROGRAMA 2

int main(){
    const char *name = "memoria";
    const int SIZE = 50261;
    int fin = SIZE;
    int inicio = 15;
    int temp;
    unsigned char nueva_imagen [SIZE];
    void *ptr;

    //abrimos el segmento
    int shm_fd = shm_open(name,O_RDONLY, 0666);

    if(shm_fd <0){
        printf("shared memory failed");
        return -1;
    }

    ptr = mmap(0,SIZE,PROT_READ, MAP_SHARED,shm_fd,0);

    if(ptr == MAP_FAILED){
        printf("Map FAILED");
        return -1;
    }

    //leemos del segmento ptr y copiamos en nueva imagen
    memcpy(&nueva_imagen,ptr,SIZE);

    while(inicio < fin){
        // Exchange
        temp = nueva_imagen[inicio];
        nueva_imagen[inicio] = nueva_imagen[fin];
        nueva_imagen[fin] = temp;
        // Get closer to middle
        inicio++;
        fin--;
    }

    int out_fd = open(NEW_FILE, O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

```

if (out_fd < 0) {
    printf("error en output");
    return -1;
}
//se escribe en cat2.pmg el contenido a partir de la dirección de memoria de nueva
imagen.
write(out_fd,&nueva_imagen[0],SIZE);

close(out_fd);

return 0;

/*nueva_imagen[0] = *ptr; // byte al que apunta
ptr++;

int *n; //se define un puntero
int a = 5;
n = &a; //n apunta a la estructura a
*n = 6; //la estructura a ahora vale 6
*/
}

```

## Ejercicio 2

Modifique su entrega de XINU que implementa el driver del teclado. Agregue un programa al shell llamado “teclado”, que verifique el uso del teclado y system call read(). El programa “teclado” debe borrar toda la pantalla gráfica, solicitar 20 pulsaciones de tecla al sistema operativo utilizando read, y mostrar los códigos hexadecimales de las 20 pulsaciones de teclas devueltas por el sistema operativo. La función que limpia la pantalla se llama paint\_screen();

```

#include <xinu.h>
extern paint_screen();

void teclado(){
    int tamanio = 20;
    unsigned char tecla;
    int i;
    char t[80];
    int inicio = 0;
    paint_screen();
    open(KEYBOARD,NULL,NULL);

    for(i = 0 ; i < tamanio; i++){
        read(KEYBOARD, &tecla,1);
        sprintf(t,"tecla 0x%x      ", tecla);
        print_text_on_vga(300, 300 + inicio, t);
        inicio = inicio + 15;
    }
    close(KEYBOARD);
    exit();
}

```



### Ejercicio 3

Responda:

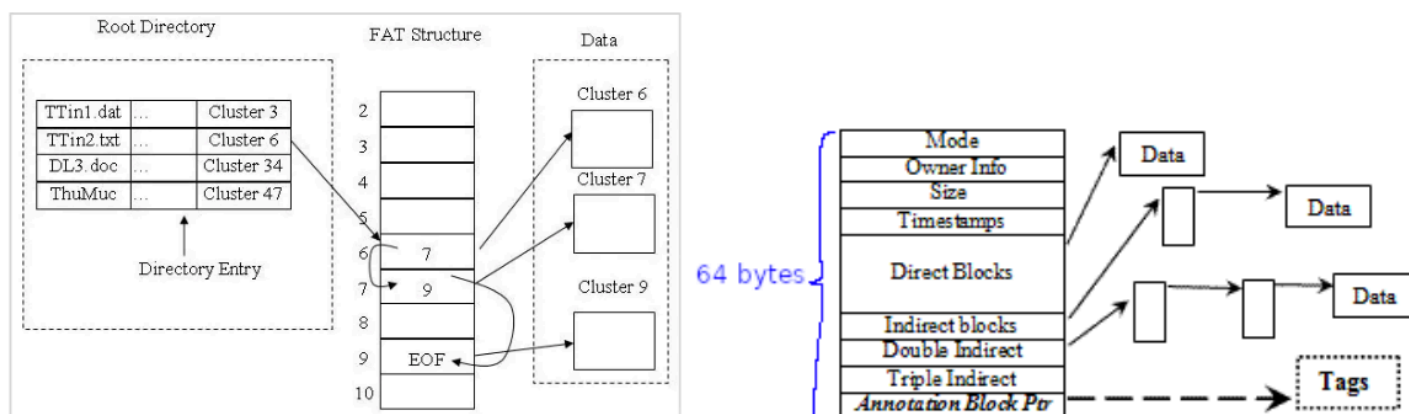
1. ¿Qué problemas podría ocasionar a su sistema XINU que un programa no utilice apropiadamente el paradigma `open()`, `read()`, `close()`?. Por ejemplo, si no se utiliza apropiadamente en la resolución del ejercicio 2. Justifique su respuesta.

Si un programa en XINU no usa correctamente `open()`, `read()`, y `close()`, puede provocar varios problemas graves. Los recursos del sistema, como los descriptores de archivos, pueden agotarse si no se cierran correctamente, causando que otros programas no puedan abrir archivos. Esto puede llevar a corrupción de datos, inconsistencias y condiciones de carrera, donde múltiples procesos interfieren entre sí, afectando el rendimiento y la estabilidad del sistema. Además, el uso inadecuado de estos recursos puede exponer el sistema a riesgos de seguridad y hacer que el rendimiento general se degrade, resultando en aplicaciones menos responsivas y posibles fallos en el sistema.

2. Es de conocimiento general que los sistemas Windows de empresas y personas sufren ataques de distintas índoles todo el tiempo. Millones de agujeros de seguridad en Windows se han reportado en internet en los últimos 30 años. Si usted desarrolla dos programas en Windows (`prog1` y `prog2`), y dos programas en XINU (`prog1` y `prog2`), responda: ¿en qué sistema es más sencillo realizar un ataque desde `prog1` a `prog2`?. Justifique su respuesta.

En Windows, es más sencillo realizar un ataque desde `prog1` a `prog2` que en XINU. Windows es un sistema operativo complejo y ampliamente utilizado, lo que lo hace un objetivo frecuente para atacantes. Con millones de vulnerabilidades reportadas a lo largo de los años, existen múltiples vectores de ataque como acceso a memoria, exploits de red y privilegios escalados que pueden ser explotados por un programa malicioso. En cambio, XINU es un sistema operativo mucho más simple y específico para entornos educativos y embebidos, con un diseño minimalista que reduce la superficie de ataque y las posibles vulnerabilidades. Además, XINU no tiene la misma complejidad y características que Windows, lo que hace que los ataques entre programas en XINU sean mucho más difíciles de llevar a cabo.

3. Sean las siguientes estructuras de dos sistemas de archivos: FAT y un FS UNIX.



- Suponga que el i-nodo de un archivo deseado ya está en RAM en el UNIX FS.
- Suponga que la FAT está en RAM.
- Suponga que en ambos sistemas el tamaño máximo para un archivo son iguales, y los bloques de datos son del mismo tamaño.

Si sólo se desea conocer los números de bloques en el disco que corresponden a los datos del archivo deseado (no su contenido), ¿en cuál de los dos sistemas de archivos sería más veloz obtener todos los números de bloques de datos que pertenecen al archivo?. Justifique.

## Segundo Parcial 2022

Ejercicio 1: Implementación del driver del teclado utilizando pasaje de mensajes.

- a. Modifique su driver de teclado para que en vez de utilizar un buffer, se utilice el mecanismo de comunicación de pasaje de mensajes. Esta implementación no utiliza buffer. Simplemente, la sección upper-half, cuando se le solicita hacer una lectura de entrada, bloqueará esperando el mensaje. Cuando suceda un evento del hw del teclado, la sección lower-half del driver enviará con un mensaje la tecla presionada. Cuando el mensaje arriba a la sección upper-half, significa que una tecla ha sido presionada, y la llamada al sistema bloqueada puede continuar y retornar a su llamador.

Para este ejercicio utilice el código original de Xinu:

<http://se.fi.uncoma.edu.ar/so/misc/xinu-pc.tar.gz>

y reemplace los archivos del driver del teclado por su driver del teclado del TP de E/S.

- b. Verifique que funcione normalmente: Implemente un programa compuesto por dos procesos.
- Un proceso lee una entrada desde el teclado ps/2 y el segundo proceso lo muestra por pantalla.
  - El proceso que lee la entrada debe entregar la lectura del teclado vía pasaje de mensajes al segundo proceso, luego, volver a esperar una entrada.
  - El segundo proceso puede mostrar en la pantalla VGA (amarilla), utilizando un código similar a este:

/\* Porción de pseudocódigo del proceso dos para mostrar en vga: \*/

int scancode;

scancode = /\* COMPLETAR esperando un mensaje del proceso 1 \*/

sprintf(t, "kbd: 0x%x ", scancode);

print\_text\_on\_vga(10, 300, t);

Otra opción es que el proceso 2, una vez que recibe el , presente con printf() sobre la tty (donde se ejecuta el shell).

```
#include <xinu.h>

int pid1, pid2;

// Proceso 1
void lector() {
    int scancode;
    open(KEYBOARD, NULL, NULL);
    while(1) {
        read(KEYBOARD, &scancode, 1);
        send(pid2, scancode);
    }
}

// Proceso 2
void mostrador() {
    int tecla;
    char t[80];
```

```

int inicio = 0;
while(1){
    tecla = receive();
    sprintf(t,"tecla 0x%x      ", tecla);
    print_text_on_vga(300, 300, t);
    inicio = inicio + 15;
}
}

void ejercicio1_2022(){

    pid1 = create(lector, 1024, 20, "proceso 1",0);
    pid2 = create(mostrador, 1024, 20, "proceso 2",0);

    resume(pid1);
    resume(pid2);

    sleep(22); // SI POR QUE ES XINU SINO CON WAITPID()

    kill(pid1);
    kill(pid2);

    close(KEYBOARD);
    exit();
}

```

## EJERCICIO 2 PERO CON MEMORIA COMPARTIDA

- Desarrollar un programa en lenguaje C en Linux, que lea el contenido del libro `principe_y_mendigo.txt`, y lo coloque en una sección de memoria del programa que previamente se solicitó dinámicamente al sistema operativo. El tamaño del archivo no se conoce a priori, por lo que el programa debe obtenerlo previo a solicitar la memoria.  
 Descargar el archivo `principe_y_mendigo.txt` usando este comando:  
`wget http://se.fi.uncoma.edu.ar/so/parciales/2do_parcial/principe_y_mendigo.txt`
- Demuestre que en la posición nro 65512 del archivo se encuentre la palabra "Catalina".  
 Una solución posible es alcanzar esa posición en el segmento de memoria con el contenido del archivo, o alcanzar esa posición mencionada en el archivo abierto (usando `lseek()` u otro). Una vez alcanzada esa posición mostrar los siguientes 8 bytes en pantalla.
- Presente las direcciones de memoria virtual de los segmentos del programa. Responda: ¿Cuáles son las direcciones del segmento que el sistema operativo le reservó a su programa dinámicamente cuando se le solicitó?. Si necesita hacer una pausa cuando el programa está en ejecución puede agregar un bucle infinito temporalmente al final del código.

## PROGRAMA 1

```

#include <stdio.h>
#include <sys/mman.h>
#include <stdlib.h> /* malloc */

```

```
#include <sys/stat.h> /* Para obtener el tamaño del archivo */
#include <unistd.h> /* Para ftruncate */
#include <fcntl.h> /* Para constantes O_* */

#define FILE "principe_y_mendigo.txt"

const char *name = "memoria";
const unsigned char *pathName = "principe_y_mendigo.txt";
void *ptr;
struct stat st;

int main()
{
    stat(pathName, &st);
    const int SIZE = st.st_size;

    // Solicitar crear segmento de memoria compartida, escritura/lectura
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    // Configurar el tamaño del segmento compartido
    ftruncate(shm_fd, SIZE);

    // Mapear segmento de memoria compartida en el espacio de dirección del proceso
    ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED){
        printf("Mapeo fallido\n");
        return -1;
    }

    // Abrimos el archivo solo para lectura
    int in_fd = open(FILE, O_RDONLY);
    if (in_fd < 0){
        printf("Error open\n");
        return -1;
    }

    // Copiamos el contenido en el segmento de memoria compartida
    if(read(in_fd, ptr, SIZE) == -1){
        printf("ERROR read\n");
        exit(1);
    }
    // Cerramos el archivo
    if (close(in_fd) < 0){
        printf("Error al cerrar el archivo");
    }

    printf("Funciono C:\n");
}
```

```
    return 0;
}
```

## PROGRAMA 2

### EJERCICIO 2B (TONY)

```
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h> /* Para ftruncate */
#include <sys/stat.h> /* Para obtener el tamaño del archivo */
#include <fcntl.h> /* Para constantes O_* */
#include <stdlib.h>
#include <string.h>
int main(){
    const char *name = "MEMORIA";
    void *ptr; //puntero
    const char *pathName = "principe_y_mendigo.txt";
    struct stat st;
    stat(pathName,&st);
    const int SIZE = st.st_size;
    int inicio = 65512;

    unsigned char texto[SIZE]; //defino el arreglo para guardar el txt
    unsigned char palabra[8]; //defino el arreglo para guardar la palabra a partir del
byte 65237.

    //solicitar seg. de mem. compartida (solo lectura)
    int shm_fd = shm_open(name,O_RDONLY, 0666);

    if(shm_fd < 0){
        printf("fallo al crear seg. mem. compartida");
        return -1;
    }
    //mapear seg. memoria compartida en el espacio de dirección del proceso.
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd,0);
    if(ptr == MAP_FAILED){
        printf("mapeo fallido\n");
        return -1;
    }

    //leer desde la mem. compartida y copiar en la dirección de memoria de texto.
    memcpy(&texto,ptr,SIZE);
    //otra forma
    /*
    for(int i = 0; i < SIZE ; i++){
        texto[i] = *ptr;
        ptr++;
    }
    */
    for(int i = 0; i < 8 ; i ++){
        palabra[i] = texto[inicio + i]; //copio los primeros 8 bytes a partir de inicio.
    }
}
```

```

printf("Palabra en 65512 y las 8 letras que le siguen: %s\n",palabra);

char cadena[] = "Catalina";
//int iguales = strcmp(cadena,palabra);
//Otra forma sin el strcmp
int j = 0;
int iguales = 1;
while(iguales && j < 8){ //mientras no cambie la bandera o el indice no llegue a 8.
    if(cadena[j] != palabra[j]){ //si no se trata de la misma palabra:
        iguales = 0; //cambia el estado
    }else{
        j++; //sino sigue verificando.
    }
}

if(iguales){
    printf("La cadena es Catalina.\n");
}else{
    printf("La cadena NO es Catalina.\n");
}

/* PARA ENCONTRAR EN QUE POSICION ESTA LA PALABRA

texto[SIZE] = '\0'; // Asegurar que la cadena esté terminada en nulo
char cadena[] = "Catalina";
// Buscar la palabra "Catalina" en el texto
unsigned char *pos = strstr(texto, cadena);

if (pos != NULL) {
    pos = pos - texto;
    printf("La cadena '%s' se encuentra en el texto en la posicion %d.\n", cadena,
pos);
} else {
    printf("La cadena '%s' NO se encuentra en el texto.\n", cadena);
}
*/

return 0;
}

```

## EJERCICIO 2A) y 2B): Alba (memoria dinámica)

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

```

```

#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h> /* Para obtener el tamaño del archivo */
#include <sys/wait.h>
#include <string.h> /* para el memcpy*/

#define ARCHIVO "principe_y_mendigo.txt"

int main(){
    int size;
    struct stat st;
    unsigned char *contenido;

    // Put attributes from file in archivo in struct st
    stat(ARCHIVO,&st);
    // Get size in bytes from struct
    size = st.st_size;

    unsigned char info[size];

    contenido = malloc(size);

    if(contenido == NULL){
        printf("ERROR malloc\n");
        return -1;
    }

    /* le pide al sistema operativo abrir el txt*/
    int in_fd = open(ARCHIVO, O_RDONLY);

    if (in_fd < 0) {
        printf("Error open");
        return -1;
    }

    //colocamos size bytes de la inf indicada por in_fd en la seccion de memoria
    if (read(in_fd, contenido, size) == -1){
        printf("ERROR read\n");
        return -1;
    }

    //inciso b
    //solucion con memcpy

    memcpy(&info,contenido,size);
    int inicio = 65512;
    char letra;

    for(int i=0;i<8;i++){

```

```

        letra = (char) info[inicio+i];

        printf("0x%x  es: %c  \n",info[inicio+i],letra);
    }

    //inciso b
    //solucion con el puntero

    printf("otra solucion \n");

    unsigned char *aux = contenido; //guardamos la posicion inicial del puntero del
segmento de memoria

    contenido = contenido + 65512; //el puntero va a la posicion 65512

    for(int j=0;j<8;j++){
        letra = (char) *contenido; //recuperamos el byte contenido
        printf("direccion: %p  es: %c  \n",(void *)contenido,letra);
        contenido++; //incrementamos la dirección del puntero
    }

    contenido = aux; //hacemos que el puntero regrese a su dirección inicial para
que luego se pueda liberar

    //continua
    close(in_fd);

    free(contenido);

    printf("Archivo txt leído y colocado en la seccion de memoria solicitada");

    return 0;

}

```

salida:



```

0x43 es: C
0x61 es: a
0x74 es: t
0x61 es: a
0x6c es: l
0x69 es: i
0x6e es: n
0x61 es: a
otra solucion
direccion: 0x7668b3566ff8 es: C
direccion: 0x7668b3566ff9 es: a
direccion: 0x7668b3566ffa es: t
direccion: 0x7668b3566ffb es: a
direccion: 0x7668b3566ffc es: l
direccion: 0x7668b3566ffd es: i
direccion: 0x7668b3566ffe es: n
direccion: 0x7668b3566fff es: a
Archivo txt leido y colocado en la seccion de memoria solicitada
alba@alba ~/E/parcial 2022>

```

## EJERCICIO 2A) y 2B): alba (memoria compartida)

```

#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h> /* Para ftruncate */
#include <fcntl.h> /* Para constantes O_* */
#include <string.h> /* para el memcpy*/
#include <sys/stat.h> /* Para obtener el tamaño del archivo */

#define ARCHIVO "principe_y_mendigo.txt"

int main(){
    const char *name = "memoria";
    int size;
    struct stat st;
    void *ptr;

    // Put attributes from file in struct st
    stat(ARCHIVO,&st);
    // Get size in bytes from struct
    size = st.st_size;

    unsigned char info[size];

    //creamos el segmento de memoria compartida
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    //configuramos el tamaño de la memoria compartida
    ftruncate(shm_fd,size);

    //now map the shared memory segment in the address space of the process

```

```

ptr = mmap(0,size, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,0);

if(ptr == MAP_FAILED){
    printf("Map FAILED");
    return -1;
}

//abrimos el txt

int in_fd = open(ARCHIVO,O_RDONLY);

if (in_fd < 0) {
    printf("Error open");
    return -1;
}

//copiamos size bytes de in_fd (el txt) en el ptr (segmento de memoria compartida)
read(in_fd,ptr,size);

//inciso b
//solucion con memcpy

memcpy(&info,ptr,size);
int inicio = 65512;
char letra;

for(int i=0;i<8;i++){

    letra = (char) info[inicio+i];

    printf("0x%x es: %c \n",info[inicio+i],letra);
}

//inciso b
//solucion con puntero

printf("otra solucion \n");

ptr = ptr + 65512; //el puntero va a la posicion 65512

for(int j=0;j<8;j++){
    letra = *((char *)ptr); //recuperamos el byte contenido
    /*
    (char *)ptr convierte el puntero ptr a un puntero de tipo char *, y luego
    *((char *)ptr) desreferencia este puntero convertido, obteniendo así el byte apuntado
    por ptr
    */
    printf("direccion: %p es: %c \n",(void *)ptr,letra);
    ptr++; //incrementamos la dirección del puntero
}

//continua

```

```
if (close(in_fd) < 0){
    printf("Error al cerrar el archivo");
}

printf("Archivo txt leído y colocado en la seccion de memoria compartida");

return 0;

}
```

## Segundo Parcial 2024

### Ejercicio 1:

```
/* Compilar con: gcc -o ordenar ordenar.c -lpthread */
```

```
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int numeros[1000];
int final[1000];

void ordenar(void *n) {
    int i, j, desde, hasta, aux;
```

```

desde = *((int *) n);
hasta = desde + 500;

/* ordenamos. metodo burbuja */
for(i = desde; i < hasta-1; i++) {
    for(j = i+1; j < hasta; j++) {
        if (numeros[i] > numeros[j]) {
            aux = numeros[i];
            numeros[i] = numeros[j];
            numeros[j] = aux;
        }
    }
}

}

void fusionar_las_dos_mitades() {
    int i, j, pos;

    i = 0;      /* indice mitad izq */
    j = 500;    /* indice mitad der */
    pos = 0;    /* indice arreglo final */

    while(i < 500 || j < 1000) {

        if (i == 500) {      /* si solo restan nros de la der */
            final[pos] = numeros[j];
            j++;
        } else if (j == 1000) { /* si solo restan nros de la izq */
            final[pos] = numeros[i];
            i++;
        } else if (numeros[i] < numeros[j]) {
            /* el numero de la primer mitad es menor */
            final[pos] = numeros[i];
            i++;
        } else {
            /* el numero de la segunda mitad es menor */
            final[pos] = numeros[j];
            j++;
        }
        pos++;
    }
}

```

```
void main() {
    int i, n, fd, t1, t2;
    pthread_t tid[2];

    /* cargamos los mil enteros */
    fd = open("desordenado.txt", O_RDONLY);
    for (i=0; i<1000; i++) {
        read(fd, &n, sizeof(int));
        numeros[i] = n;
    }
    close(fd);

    t1 = 0;
    t2 = 500; /* segundo thread desde el indice 500 */

    pthread_create(&tid[0], NULL, (void*)ordenar, (void *)&t1);
    pthread_create(&tid[1], NULL, (void*)ordenar, (void *)&t2);

    /* esperamos a que los threads finalicen */
    for (int i = 0; i <= 1; i++) {
        pthread_join(tid[i], NULL);
    }

    fusionar_las_dos_mitades();

    /* mostramos el arreglo final */
    for(int i = 0; i < 1000; i++) {
        printf("%i ", final[i]);
    }
}
```

## Ejercicio 2:

```
#include <xinu.h>

#define NO_ES_PRIMO 0
#define ES_PRIMO 1
#define NO_ES_BISIESTO 2
#define ES_BISIESTO 3

int es_primo(int);

/* proceso que verifica si es primo */
void proceso2(int padre) {
    int n;

    n = receive();

    if (es_primo(n))
        send(padre, ES_PRIMO);
    else
        send(padre, NO_ES_PRIMO);

    /*
     * evitar por 10 seg que este proceso finalice, para que XINU
     * no envíe un msg al padre de que proceso2 finalizó
     */
    sleep(10);
}

/* proceso que verifica si es bisiesto */
void proceso3(int padre) {
    int anio;

    anio = receive();
    sleep(5); /* esperamos, para evitar que el padre pierda algun msg */

    if ((anio % 4 == 0) && ((anio % 100 != 0) || (anio % 400 == 0)))
        send(padre, ES_BISIESTO);
    else
        send(padre, NO_ES_BISIESTO);
}

void mostrar_resultado(int msj) {

    if (msj==ES_BISIESTO) {
        printf ("el numero ingresado es bisiesto\n");
    } else if (msj==NO_ES_BISIESTO) {
        printf ("el numero ingresado NO es bisiesto\n");
    }
}
```

```

    } else if (msj==ES_PRIMO) {
        printf ("el numero ingresado es primo\n");
    } else if (msj==NO_ES_PRIMO) {
        printf ("el numero ingresado NO es primo\n");
    }
}

void ej2_parcial(){
    int n, pid, proc2, proc3, i;
    int msj;
    char buf[10];

    /* Solicitamos un nro al usuario */
    printf("INGRESE UN NRO: ");
    read(CONSOLE, buf, 10);
    n = atoi(buf);
    printf ("el numero ingresado entero es %d \n", n);

    pid = getpid();

    proc2 = create(proceso2, 8192, 20, "Proceso 1", 1, pid);
    proc3 = create(proceso3, 8192, 20, "Proceso 2", 1, pid);

    resume(proc2);
    resume(proc3);

    /* Enviamos el nro del usuario a los procesos */
    send(proc2, n);
    send(proc3, n);

    msj= receive();
    mostrar_resultado(msj);
    msj= receive();
    mostrar_resultado(msj);
}

int es_primo(int num) {
    int res, i;

    res = 1;    /* es primo */
    i = 2;
    while ((i<num) && (res)) {

        if ((num%i) == 0)
            res = 0;    /* no es primo */
        i++;
    }
}

```

```
    return res;  
}
```

