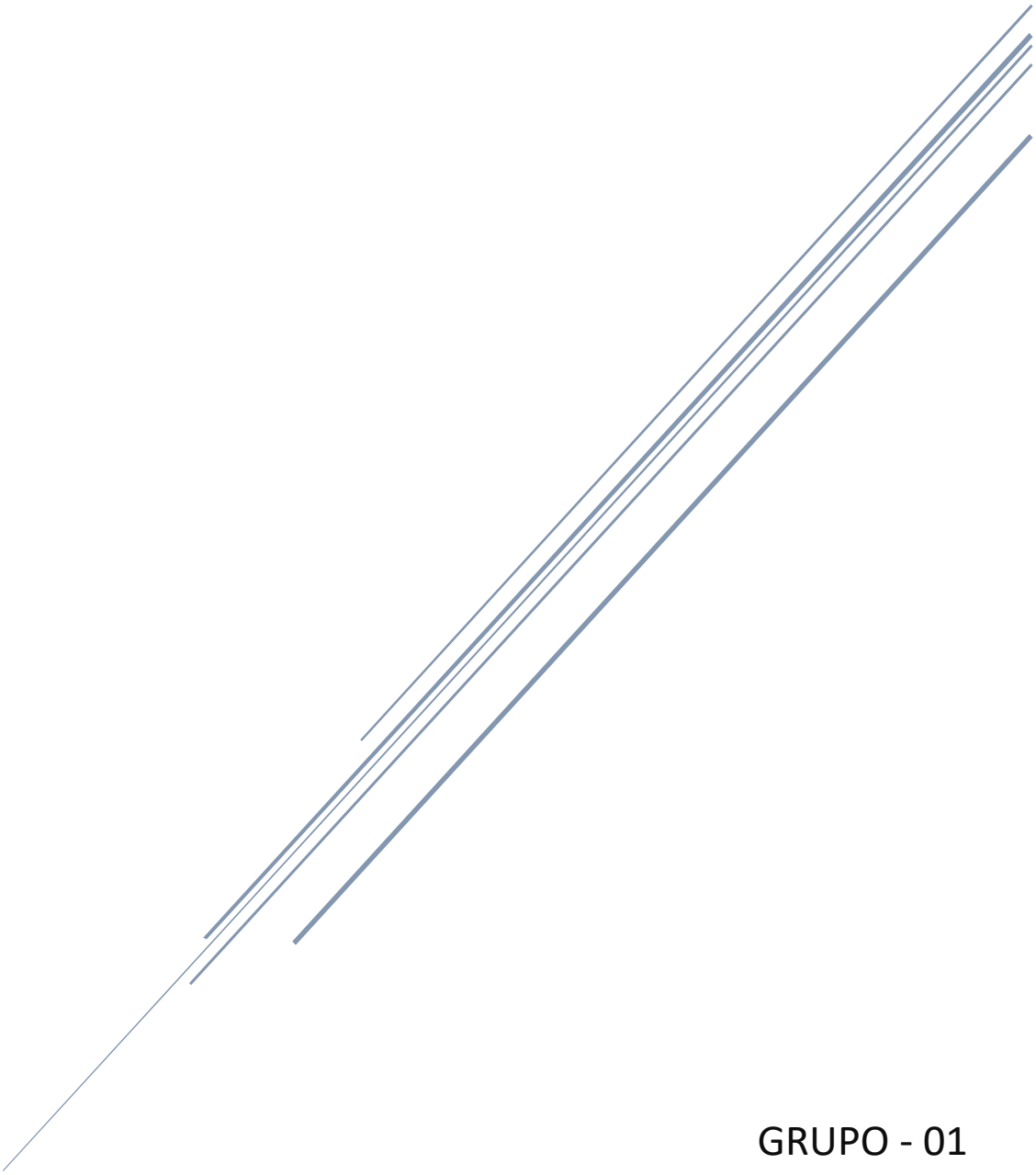
# BATRACIOS

## Práctica UNIX

GRADO INGENIERIA INFORMÁTICA EN SISTEMAS DE INFORMACIÓN

Sistemas Operativos II

2020/2021



GRUPO - 01
David Barrios Portales
GRUPO - 01

# Contenido

# 1 – Recursos IPC usados

## 1.1 - Semáforos

Vamos a usar un conjunto de semáforos, que se va a almacenar su ID en la variable global *id_semaforo*, en concreto estará formado por 9 semáforos.

Cuatro de ellos serán para el control de las ranas madre, otros tres restantes serán para llevar el conteo de las ranas nacidas, salvadas y perdidas, otro semáforo para las posiciones de cada ranita hija y por último un semáforo para controlar que haya un número máximo de 35 procesos, 1 proceso "general", 4 ranas madre y 30 ranitas. Este último número es el que vamos a controlar, 30, que está almacenado en la constante MAX_RANAS_HIJAS.

La declaración de los semáforos es la siguiente en la función main:

```
id_semaforo = semget(IPC_PRIVATE, 10, IPC_CREAT | 0600);
```

En esta línea hemos declarado un conjunto de semáforos, más en concreto 10 semáforos, hemos almacenado el id del conjunto en la variable id_semaforo.

Más adelante establecemos el máximo de "accesos" que permite cada semáforo. Un ejemplo

```
semctl(id_semaforo, RANA_MADRE_1, SETVAL, 1);
semctl(id_semaforo, RANA_MADRE_2, SETVAL, 1);
semctl(id_semaforo, RANA_MADRE_3, SETVAL, 1);
semctl(id_semaforo, RANA_MADRE_4, SETVAL, 1);
```

con los semáforos que controla cada rana madre:

Las definiciones de cada constante son:

```
#define RANA_MADRE_1 2          // De la primera rana madre
#define RANA_MADRE_2 3          // De la segunda rana madre
#define RANA_MADRE_3 4          // De la tercera rana madre
#define RANA_MADRE_4 5          // De la cuarta rana madre
```

Funcionamiento del semáforo de MAX_RANAS_HIJAS:

Sólo se pueden tener un máximo de 30 ranas hija en pantalla, por lo tanto, el semáforo tiene 30 posiciones, las cuales, cuando se crea una nueva rana hija, se decrementa el contador, cuando muere o se salva se incrementa, para así dejar "hueco" para una nueva ranita.

También tenemos semáforos para acceder a la memoria compartida, donde se almacenan las posiciones de las ranas y también las ranas nacidas, salvadas y muertas.

Estos semáforos ocupan el numero SEMAF_RANITAS_NACIDAS, SEMAF_RANITAS_SALVADAS, SEMAF_RANITAS_MUERTAS y SEMAF_POSICIONES dentro del conjunto de semáforos.

SEMAF_POSICIONES controla el acceso a la memoria compartida, para que cuando se esta modificando una posición de una ranita, no se pueda acceder a ella de ninguna otra forma.

## 1.2 Memoria compartida

Vamos a crear los 2048xint necesarios para el funcionamiento correcto de la biblioteca proporcionada. El id de esta memoria se guardará en id_memoria.

También vamos a usar 33 posiciones de una estructura que hemos llamado “posicion_struct” (modificando el nombre respecto al incluido en la biblioteca, ya que el nombre era más confuso).

La declaración de la estructura es la siguiente:

```
struct posicion_struct {int x,y;};
```

Posicionamiento de los datos:

- 0-29: Almacena las posiciones X e Y de cada ranita hija.
- 30-32: Almacena un contador de las ranas nacidas, salvadas y perdidas, respectivamente. De esta manera podemos llevar fácilmente el conteo de nuestras ranitas.

Una vez hecho esto enlazamos la memoria compartida con las variables posiciones y memoria, así:

```
memoria = (char *)shmat( id_memoria, NULL, 0);
posiciones =(struct posicion_struct *) shmat(id_posiciones, NULL, 0);
```

Inicializamos todas las posiciones de memoria a -2, ambas componentes.

Como última tarea antes de comenzar, ponemos a 0 tanto el contador para ranas nacidas, el de ranas salvadas y ranas perdidas de la siguiente forma:

```
//La memoria para las ranitas nacidas
semaforo_wait(id_semaforo,SEMAF_RANITAS_NACIDAS);
//Lo ponemos a 0 al principio
posiciones[30].x=0;
semaforo_signal(id_semaforo,SEMAF_RANITAS_NACIDAS);
```

Vemos si podemos acceder a la memoria compartida de las posiciones, si el semáforo nos los permite entonces ponemos a 0 dicho contador y cuando acabamos damos un signal al semáforo.

Así con el resto de contadores.

### 1.3 Señales

En la ejecución se enmascara la señal “SIGINT”, para que cuando se mande esta señal usando “CTRL^C”, la ejecución del programa acabe.

## 2 - Variables globales

Las variables globales que hemos usado son: *id_semaforo*, un puntero a finalizar y por último una estructura para almacenar las posiciones.

- La variable *id_semaforo* es usada para, como su nombre indica, almacena el ID del semáforo usado en la práctica.
- En la variable *finalizar* vamos a almacenar el estado, para en los “bucles infinitos” poder salir
- La variable *id_memoria* contiene el ID de la memoria compartida.

## 3 – Pseudocódigo

### Función ACABAR / MANEJADORA de la señal SIGINT

```
Función ACABAR devuelve vacio
        parámetros
                int s
        inicio
                Si s = SIGINT entonces
                        Variable Puntero finalizar = 1
                        Variable global_control = 0
                Fin si
        Fin
```

Fin

## Función SEMAFORO_WAIT

```
Funci ó n  SEMAFORO_WAIT devuelve entero
        Par á metros
                Sem á foro_id entero
                índice  Entero
        Variables
                Estructura sembuf oper
        Inicio
                Oper.numero sem á foros =  í ndice
                Oper.selecci ó n operaci ó n sem á foro = -1
                Oper.flags sem á foro = 0
        devolver semop (semafoto_id, &oper, 1)
```

## Función SEMAFORO_SIGNAL

```
Funci ó n  SEMAFORO_SIGNAL devuelve entero
        Par á metros
                Sem á foro_id entero
                índice  Entero
        Variables
                Estructura sembuf oper
        Inicio
                Oper.numero sem á foros =  í ndice
                Oper.selecci ó n operaci ó n sem á foro = +1
                Oper.flags sem á foro = 0
        devolver semop (semafoto_id, &oper, 1)
```

## Función RANITA

```
Funci ó n RANITA devuelve entero
        Parametros
                Entero pos, i
        Inicio
                // Mientras no se pueda saltar y no hayamos acabado, se queda en el bucle
                Mientras  BATR_puedo_saltar (posiciones[pos.x, posiciones[pos].y, arriba) ≠ 0
y  *finalizar ≠ falso

                Si BATR_puedo_saltar (posiciones[pos].x, posiciones[pos].y, ARRIBA) = 0
entonces
                        BATR_avance_rana_ini (posiciones[pos].x,posiciones[pos].y)
                        BATR_avance_rana (&posiciones[pos].x,&posiciones[pos].y,ARRIBA)
                        BATR_avance_rana_fin (posiciones[pos].x,posiciones[pos].y)
                Fin si
```

```
                //llamada a la funci ó n sem á foro_signal
                sem á foro_signal (id_semaforo, (i+2))

                //se avanza mientras finalizar no sea falso
mientras *finalizar ≠ falso entonces
                        si sem á foro_wait (id_semaforo, SEMAF_POSICIONES = -1 entonces
                                continua
                        fin si

                //Comprobacion para ver si la rana ha cruzado
                Si posiciones[pos].y = RANA_CRUZADA entonces
                        semaforo_wait (id_semaforo, SEMAF_RANITAS_SALVADAS)
                        posiciones[31].x++;     //se incrementa ranas salvadas
                        semaforo_signal (id_semaforo,SEMAF_RANITAS_SALVADAS)
                        //Ponemos a -2 para saber que esa rana esta libre
                        posiciones[pos].x =-2
                        posiciones[pos].y = -2
                        semaforo_signal (id_semaforo, MAIN_PANTALLA)

                        semaforo_signal (id_semaforo, SEMAF_POSICIONES)
                        devolver 0
                fin si

                //comprobaci ó n de que la x se encuentra entre 0 y 80
                sino  posiciones[pos].x <= 0  ó  posiciones[pos].x >= 80 entonces
                        posiciones[32].x++      // aumenta contador ranas muertas
```

```
                posiciones[pos].x = -2
                posiciones[pos].y = -2
                semaforo_signal (id_semaforo, MAIN_PANTALLA)

                semaforo_signal (id_semaforo, SEMAF_POSICIONES)
                devolver 0
            fin sino

            //sino para si puede saltar
            sino BATR_puedo_saltar (posiciones[pos].x, posiciones[pos].y, ARRIBA) = 0
                BATR_avance_rana_ini (posiciones[pos].x, posiciones[pos].y)
                BATR_avance_rana (&posiciones[pos].x, &posiciones[pos].y, ARRIBA)
                BATR_pausa
                BATR_avance_rana_fin(posiciones[pos].x,posiciones[pos].y)

                semaforo_signal(id_semaforo, SEMAF_POSICIONES)
fin mientras

si global_control = 0 y posiciones[pos].y > -1 y posiciones[pos].y < 11 y posiciones[pos].x < 80
y posiciones[pos].x > 0 entonces
    //explotamos la rana que no ha llegado arriba o ha salido por uno de los lados
    BATR_explotar(posiciones[pos].x,posiciones[pos].y
    //incrementa contador ranas muertas
    posiciones[32].x++
    devolver 0
fin si
devolver 0
fin funci ó n ranita
```

## Función RANA MADRE

```
Funci ó n c ó digo_rana_madre devuelve entero
    Par á metros
        Entero i
    Variables
        Entero x, y, posici ó n, j, k
        Pid_t id_ranita
    Inicio
        Mientras *finalizar ≠ falso //bucle infinito
            BATR_descansar_criar //ranas madre descansan
            //wait para esperar a un hueco para nueva ranita
            Si semaforo_wait (id_semaforo, MAIN_PANTALLA) = -1 entonces
                Continua
            Fin si
            //wait para ver si rana anterior se ha movido
            Si semaforo_wait(id_semaforo,(i+2)) = -1 entonces
                Devuelve 0
            Fin si
            Si semaforo_wait (id_semaforo, SEMAF_POSICIONES) = -1 entonces
                Continua
            Fin si
            Para j = 0 con j <= 29 y ++j hacer
                Si posiciones[j].x = -2 entonces
                    Posici ó n = j
                    Romper
                Fin si
            Fin para
```

```
                    Si global_control = 1 entonces
                        //llamada a la funci ó n que crea ranitas
                        BATR_parto_ranas(i, &posiciones[posicion].x,
&posiciones[posicion].y);
                        semaforo_signal (id_semaforo, SEMAF_POSICIONES)
                        semaforo_wait(id_semaforo,SEMAF_RANITAS_NACIDAS)
                        //se incrementa contador de ranas nacidas
                        posiciones[30].x++
                        semaforo_signal(id_semaforo,SEMAF_RANITAS_NACIDAS)
                        id_ranita=fork() // se crea proceso que se encargue de la ranita
indic á ndosele posici ó n y numero de madre
                    fin si
                    si no
                        devolver 0
                    fin si no
                    selector (id_ranita) ********
                        caso -1 :
                            imprimir perror("ERROR. Fork")
                            devolver 1
                        caso 0:         //c ó digo del hijo
                            devolver ranita (posicion, i)
                    fin selector
        fin ***
        devolver 0
fin funci ó n rana madre
```

### Función GENERA_ALEATORIO

Funci ó n genera_aleatorio devuelve vacio

Par á metros

Entero *vector

entero Num

Variables

Entero i

Inicio

Si num > 1 entonces

Para i = 0, i < num, i++ entonces

vector[i] = rand() % (RANDOM_MAX + 1 - RANDOM_MIN) + RANDOM_MIN

fin para

fin si

fin funci ó n genera_aleatorio

## 4 - Otras funciones relevantes

Tenemos las funciones *presentacio()* y *error_parametros(),* las cuales nos muestran una pequeña cabecera al inicio del programa y nos muestra lo que hacer si no hemos introducido los parámetros correctamente, respectivamente.

Al inicio, la práctica se pausa 7 segundos para que podamos ver por pantalla todos los datos que se nos muestran. Esto se hace creando un hijo y esperando a que finalice.

La función GENERA_ALEATORIO recibe un puntero y un entero, genera un array aleatorio a ese puntero de la dimensión que le hayamos pasado como segundo parámetro.

```
// ------------------------------------------------------------------
// Funcion genera_aleatorio
// Genera los elementos de un string de forma aleatoria
// ----------------------------------
void genera_aleatorio(int *vector,int num){
    int i;

    if(num>1){
        //Recorremos todo el vector
        for(i=0; i<num;i++){
            //E introducimos los nums aleatorios
```

```
            vector[i] = rand() % (RANDOM_MAX + 1 - RANDOM_MIN) + RANDOM_MIN;
        }
        //RANDOM_MAX
        //RANDOM_MIN
        //rand() % (max_number + 1 - minimum_number) + minimum_number
    }
}
// ------------------------------------------------------------------------
```

Se ha intentado reutilizar el máximo código de las prácticas, adaptando ciertas cosas.

Como último aporte, el código se encuentra en github.com, en el siguiente enlace:

github.com/Biohazard86/batracios_SO2

El uso de esta plataforma es muy cómodo, ya que se sube el código, se tienen diferentes versiones, por si algo que acabas de implementar no funciona, puedes volver hacia atrás. También es más cómodo porque puedes ver las aportaciones, que se ha añadido, etc.