**Batracios**

Práctica UNIX

GRUPO - 01

David Barrios Portales

GRUPO - 01

GRADO INGENIERIA INFORMÁTICA EN SISTEMAS DE INFORMACIÓN

Sistemas Operativos II

2020/2021

Contenido

[1 – Recursos IPC usados 2](#_Toc73896793)

[1.1 - Semáforos 2](#_Toc73896794)

[1.2 Memoria compartida 3](#_Toc73896795)

[1.3 Señales 4](#_Toc73896796)

[2 - Variables globales 4](#_Toc73896797)

[3 – Pseudocódigo 4](#_Toc73896798)

[Función ACABAR / MANEJADORA de la señal SIGINT 4](#_Toc73896799)

[Función SEMAFORO\_WAIT 5](#_Toc73896800)

[Función **¡Error! Marcador no definido.**](#_Toc73896801)

[Función SEMAFORO\_SIGNAL 5](#_Toc73896802)

[Función RANITA 6](#_Toc73896803)

[Función RANA MADRE 9](#_Toc73896804)

[Función GENERA\_ALEATORIO 11](#_Toc73896805)

[4 - Otras funciones relevantes 11](#_Toc73896806)

# 1 – Recursos IPC usados

## - 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 con los semáforos que controla cada rana madre:

|  |
| --- |
| 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);

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.

## 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.

## Señales

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

# - 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.

# – Pseudocódigo

## Función ACABAR / MANEJADORA de la señal SIGINT

Fin

Función ACABAR devuelve vacio

parámetros

int s

inicio

Si s = SIGINT entonces

Variable Puntero finalizar = 1

Variable global\_control = 0

Fin si

Fin

## Función SEMAFORO\_WAIT

Función SEMAFORO\_WAIT devuelve entero

Parámetros

Semáforo\_id entero

Índice Entero

Variables

Estructura sembuf oper

Inicio

Oper.numero semáforos = índice

Oper.selección operación semáforo = -1

Oper.flags semáforo = 0

devolver semop (semafoto\_id, &oper, 1)

## Función SEMAFORO\_SIGNAL

Función SEMAFORO\_SIGNAL devuelve entero

Parámetros

Semáforo\_id entero

Índice Entero

Variables

Estructura sembuf oper

Inicio

Oper.numero semáforos = índice

Oper.selección operación semáforo = +1

Oper.flags semáforo = 0

devolver semop (semafoto\_id, &oper, 1)

## Función RANITA

Función 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 función semáforo\_signal

semáforo\_signal (id\_semaforo, (i+2))

//se avanza mientras finalizar no sea falso

mientras \*finalizar ≠ falso entonces

si semáforo\_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

//comprobación 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 función ranita

## Función RANA MADRE

Función código\_rana\_madre devuelve entero

Parámetros

Entero i

Variables

Entero x, y, posición, 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

Posición = j

Romper

Fin si

Fin para

Si global\_control = 1 entonces

//llamada a la función 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 indicándosele posición 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: //código del hijo

devolver ranita (posicion, i)

fin selector

fin \*\*\*

devolver 0

fin función rana madre

## Función GENERA\_ALEATORIO

Función genera\_aleatorio devuelve vacio

Parámetros

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 función 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](https://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.