**Batracios**

Práctica UNIX

David Barrios Portales

Víctor Vacas Amigo

GRADO INGENIERIA INFORMÁTICA EN SISTEMAS DE INFORMACIÓN

Sistemas Operativos II

2020/2021

Contenido

[1 – Recursos IPC usados 2](#_Toc73569915)

[1.1 - Semáforos 2](#_Toc73569916)

[1.2 Memoria compartida 2](#_Toc73569917)

[1.3 Señales 2](#_Toc73569918)

[2 - Variables globales 2](#_Toc73569919)

[3 – Pseudocódigo 3](#_Toc73569920)

[Operación SEMAFORO\_WAIT 3](#_Toc73569921)

[Función SEMAFORO\_SIGNAL 3](#_Toc73569922)

[Función RANITA 3](#_Toc73569923)

[Función CODIGO\_RANA\_MADRE 3](#_Toc73569924)

[Función MAIN (Simplificada) 3](#_Toc73569925)

[4 - Otras funciones relevantes 3](#_Toc73569926)

# 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 10 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:

Funcionamiento del semáforo de las ranas madre:

Funcionamiento del semáforo de las ranitas:

Funcionamiento del semáforo de MAX\_RANAS\_HIJAS:

## Memoria compartida

Vamos a crear los 2048xint necesarios para el funcionamiento correcto de la biblioteca proporcionada.

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.

Una vez hecho esto enlazamos la memoria compartida con las variables posiciones y memoria.

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

## 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

## Operación SEMAFORO\_WAIT

## Función SEMAFORO\_SIGNAL

## Función RANITA

## Función CODIGO\_RANA\_MADRE

## Función MAIN (Simplificada)

# 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 6 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)