Guía práctica de Comunicación, Concurrencia y Sincronización

1. Utilizando las llamadas al sistema de *Memoria Compartida* (`shmget()`, `shmctl()`, `shmat()` y `shmdt()`) escriba dos programas **`Emisor`** y **`Receptor`** que, compartiendo un espacio de memoria, permitan que:

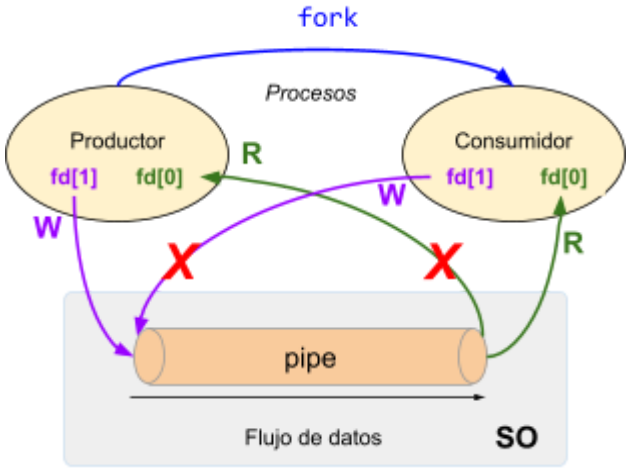
    a. el `Emisor` pueda generar un vector de 10 números aleatorios y almacenarlo en la memoria compartida.

    b. el `Receptor` pueda acceder a la memoria compartida y mostrar los valores almacenados.

2. Modifique los programas anteriores para que un proceso (**`Emisor`**) espere el ingreso por teclado y el segundo proceso (**`Receptor`**) lo escriba en pantalla.

3. Utilice los comandos `ipcs` e `ipcrm` para consultar y eliminar el segmento de memoria compartida de los programas anteriores.

4. Realice un programa de *Comunicación Unidereccional*, que mediante `pipe`, permita a un proceso padre generar una cadena de texto y enviarla al hijo, este último debe escribirla en pantalla.



5. Modifique el ejercicio anterior para permitir una *Comunicación Bidireccional* que permita a un proceso padre generar una cadena de texto y enviarla al hijo el cual debe escribirla en pantalla y luego el proceso hijo debe generar una cadena de texto y enviarla al padre el cual debe escribirla en pantalla.

6. Realice dos programas en los cuales, utilizando *Tuberías con Nombre* (`mkfifo`), permitan a un programa recibir una cadena de texto por pantalla y escribir en la tubería y al otro programa leer de la tubería y mostrar la cadena ingresada.

7. Realice dos programas en los cuales, utilizando *Colas de Mensajes*, permitan a un programa recibir una cadena de texto por consola y escribir en la cola y al otro programa leer de la cola y mostrar la cadena ingresada.

8. A partir del ejemplo de teoría que produce la secuencia de salida `ABCABCABC` repetidamente:

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <ctime>
#include <cstdlib>
#include <unistd.h>
#include <stdio.h>
#include <wait.h>

using namespace std;

sem_t sem_A, sem_B, sem_C;

void *A(void *)
{
    while (1)
    {
       sem_wait(&sem_A);
       cout << "A";
       sem_post(&sem_B);
    }
}

void *B(void *)
{
    while (1)
    {
       sem_wait(&sem_B);
       cout << "B";
       sem_post(&sem_C);
    }
}

void *C(void *)
{
    while (1)
```

```
    {
       sem_wait(&sem_C);
       cout << "C";
       sem_post(&sem_A);
    }
}

int main(int argc, char *argv[])
{
    pthread_t h1, h2, h3;

    sem_init(&sem_A, 0, 1);
    sem_init(&sem_B, 0, 0);
    sem_init(&sem_C, 0, 0);

    pthread_create(&h1, NULL, A, NULL);
    pthread_create(&h2, NULL, B, NULL);
    pthread_create(&h3, NULL, C, NULL);

    pthread_join(h1, NULL);
    pthread_join(h2, NULL);
    pthread_join(h3, NULL);

    sem_destroy(&sem_A);
    sem_destroy(&sem_B);
    sem_destroy(&sem_C);

    return 0;
}
```

   a. Modifique su código para que produzca la secuencia de salida `BACBACBAC` repetidamente.

   b. Modifique el código anterior de manera tal que ahora la secuencia de salida sea `BACA`

9. Realice un programa que calcule la operatoria de multiplicación de los números que se encuentren entre dos números enteros que se leen por teclado y cree dos procesos ligeros que se estarán ejecutando concurrentemente en el sistema. Cada proceso deberá realizar la siguiente tarea:

   a. Realizará la operatoria de multiplicación de los enteros comprendidos en la primera mitad del intervalo (entre `ini` y `fin-ini/2`). El resultado parcial obtenido lo acumulará sobre la variable global operación.

b. Realizará la operatoria de multiplicación de los enteros comprendidos en la segunda mitad del intervalo (entre `(fin-ini/2)+1` y `fin`). El resultado parcial obtenido lo acumulará sobre la variable global operación.

La hebra principal esperará a que acaben cada uno de los procesos creados (hilos) e imprimirá la variable global operación. *El control del acceso a la sección crítica se deberá realizar mediante semáforos*.

10. Realice un programa que cree dos hilos llamados `H1` y `H2` (procesos concurrentes). Cada proceso debe realizar la siguiente tarea:

    a. El proceso padre creará los dos hilos y debe imprimir por pantalla los números del 11 al 15.

    b. `H1` imprimirá por pantalla los números del 0 al 5 y finalizará su ejecución.

    c. `H2` imprimirá por pantalla los números del 6 al 10 y finalizará su ejecución.

    Los números deberán imprimirse en orden creciente esperando un valor al azar entre 0 y 1 con `sleep(rand()%2);` antes de mostrarse. *La sincronización necesaria entre los dos procesos se realizará mediante semáforos*.

11. Realice un programa que cree dos hilos llamados `H1` y `H2` (procesos concurrentes). Cada proceso debe realizar la siguiente tarea:

    a. El proceso padre creará los dos hilos y debe imprimir por pantalla los números del 41 al 50.

    b. `H1` imprimirá por pantalla los números del 0 al 10 y del 21 al 30 y finalizará su ejecución.

    c. `H2` imprimirá por pantalla los números del 10 al 20 y del 31 al 40 y finalizará su ejecución.

    Los números deberán imprimirse en orden creciente esperando un valor al azar entre 0 y 1 con `sleep(rand()%2);`. *La sincronización necesaria entre los dos procesos se realizará mediante semáforos*.

12. Considerando el siguiente programa que produce valores y calcula e imprime su doble:

```
const int num_iter = 100;       // Número de iteraciones
const int tam_vector = 5;       // Tamaño del vector
int vector[tam_vector];         // Vector para producir o consumir

int pos_productor = 0;
int pos_consumidor = 0;
```

```
int doble;

int producir_dato()
{
    static int dato = 0;
    dato++;
    return dato;
}

void duplicar_dato(int dato)
{
    doble = dato * 2;
    cout << "Dato duplicado: " << doble;
}

void *productor(void *p)
{
    for (unsigned long i = 0; i < num_iter; i++)
    {
       int dato = producir_dato(); // Produce un dato
       vector[pos_productor % tam_vector] = dato;
       pos_productor++;
       cout << "Dato producido: " << dato;
    }
    return NULL;
}

void *consumidor(void *p)
{
    for (unsigned long i = 0; i < num_iter; i++)
    {
       // Consume el valor generado
       int dato = vector[pos_consumidor % tam_vector];
       pos_consumidor++;
       duplicar_dato(dato);
    }
    return NULL;
}

int main(int argc, char **argv)
{
    pthread_t h1, h2;

    pthread_create(&h1, NULL, productor, NULL);
    pthread_create(&h2, NULL, consumidor, NULL);

    pthread_join(h1, NULL);
```

```
    pthread_join(h2, NULL);

    return 0;
}
```

Modifique el programa para que, mediante la sincronización con semáforos, permita controlar la secuencia y el acceso a sección crítica.

13. Realice un algoritmo productor consumidor que trabaje con un recurso compartido (arreglo de 5 enteros), en donde el productor irá almacenando valores al azar entre 0 y 99 y el consumidor los estará extrayendo y mostrando en la pantalla. Tenga en cuenta en controlar el acceso a la sección crítica y de tener un semáforo para el vector lleno y el vector vacío.