

**UVM****Universidad
del Valle de México**
LAUREATE INTERNATIONAL UNIVERSITIES*

MATERIA

Sistemas Operativos

ACTIVIDAD 4: Proyecto Integrador Etapa 1

UNIDAD 3: Análisis de procesos

EQUIPO: Windows 4

Ángel Alejandro Alcalá Aguilar

Romel Arenas Jiménez

Ángel Christian López

Adair Espinosa Estrada

José Emiliano Jauregui Guzmán

NOMBRE DEL DOCENTE: Manuel Triana Vega

FECHA: lunes 7 de abril de 2025

Introducción

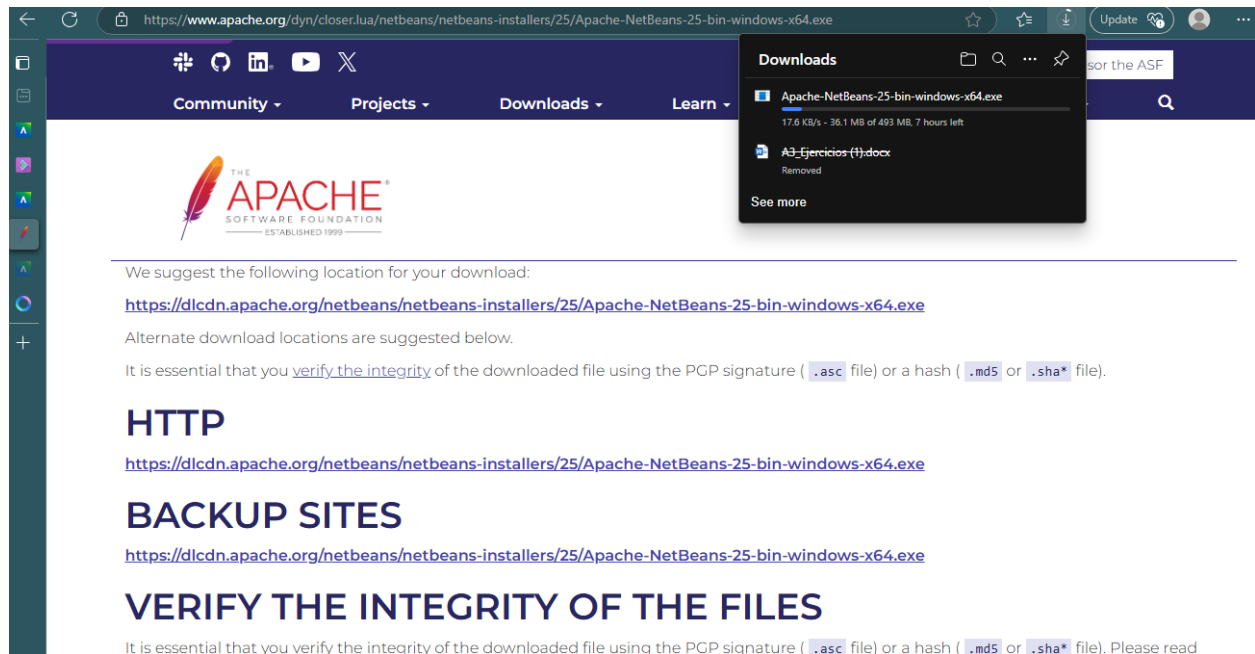
En el contexto de la asignatura de Sistemas Operativos, nuestro equipo se propone desarrollar un proyecto integrador que aplique los conceptos fundamentales de la programación para el manejo de la concurrencia y la seguridad en un sistema operativo de código abierto, específicamente Linux. A lo largo de este proyecto, trabajaremos en colaboración para diseñar y desarrollar algoritmos que permitan gestionar la concurrencia de manera efectiva y segura. En esta primera etapa, nos enfocaremos en la instalación de un entorno de desarrollo integrado (IDE) y la resolución de ejercicios relacionados con hilos y concurrencia. Nuestro objetivo es adquirir una comprensión profunda de los conceptos y técnicas involucradas en el manejo de la concurrencia y la seguridad en sistemas operativos. A través de este proyecto, esperamos desarrollar habilidades y conocimientos prácticos que nos permitan enfrentar desafíos en el campo de la tecnología y la informática.

Indicé de contenido

Etapas del Proyecto integrador

Introducción.....	2
I. Hilos y concurrencia	
1.1 Instalación de IDE.....	3
1.2 Resolución de ejercicio.....	5
Conclusiones.....	8
Referencias.....	10

1.1 Instalación de IDE



1.2 Resolución de ejercicio

- Utiliza el código que se proporciona en el documento: Java La Cena de los Filósofos.
- Compila y corre el programa en Java que resuelve el problema de la cena de los filósofos utilizando hilos y concurrencia.
- Recupera las pruebas del programa destacando las posibles modificaciones al código para que corra adecuadamente.

Clases:

Clase filosofo es la clase principal que extiende de Thread

Clase tenedor que no está implementada en el código y se tuvo que agregar.

Se utilizaron los metodos importantes para la separacion de acciones como:

Se utiliza sleep(tiempo) para pausar el hilo durante un tiempo.

Pensar():, Comer():, TomarTenedorIzq() y TomarTenedorDer():, Esperar():, Saciado():, SoltarTenedores():.

Puntos a destacar:

Al no tener una nueva interaccion entre los tenedores y los filosofos, se definen las acciones de tomas o dejar el tenedor para evitar bloques.

Evitamos problemas de concurrencia con synchronized para las acciones de comer y esperar.

```
public class Tenedor {  
    private boolean estado; // Estado de si el tenedor está disponible o no.  
  
    public Tenedor() {  
        this.estado = true; // El tenedor comienza disponible.  
    }  
  
    public synchronized boolean tomar() {  
        if (estado) {  
            estado = false; // El tenedor ahora está en uso.  
            return true;  
        }  
        return false; // El tenedor ya está en uso.  
    }  
  
    public synchronized void soltar() {  
        estado = true; // El tenedor se libera.  
    }  
}
```

```
public class Filosofo extends Thread {  
    private int id;  
    private Tenedor left, right;  
    private String estado;  
    private int tiempo;  
    int platillos;  
    int tanda = 0;  
  
    public Filosofo(int id, Tenedor left, Tenedor right) {  
        this.id = id;  
        this.left = left;  
        this.right = right;  
    }  
}
```

```

    platillos = (int) (Math.random() * 5 + 5);
    estado = "pensando";
    tiempo = (int) (Math.random() * 7 + 3) * 1000;
}

public void run() {
    while (true) {
        try {
            tanda++;
            Pensar();
        } catch (InterruptedException ex) {
            System.err.println("Error: " + ex.getMessage());
        }
    }
}

public synchronized void Pensar() throws InterruptedException {
    estado = "pensando";
    System.out.println("Filósofo " + id + " pensando...");
    sleep(tiempo); // El filósofo piensa durante un tiempo aleatorio
    Comer();
}

public synchronized void Comer() throws InterruptedException {
    System.out.println("Filósofo " + id + " intentando comer...");
    while (!TomarTenedorIzq() || !TomarTenedorDer()) {
        SoltarTenedores();
        Esperar();
    }

    estado = "comiendo";
    System.out.println("Filósofo " + id + " está comiendo el platillo " + platillos);
    sleep(tiempo); // El filósofo come durante un tiempo aleatorio
    platillos--;

    SoltarTenedores();

    if (platillos == 0) {

```

```

        Saciado();
        finishFiloso();
    } else {
        Pensar();
    }
}

public synchronized void Saciado() {
    estado = "saciado";
    System.out.println("Filósofo " + id + " está " + estado + ".");
}

public synchronized void finishFiloso() {
    try {
        this.wait();
    } catch (InterruptedException ex) {
        System.err.println("Error: " + ex.getMessage());
    }
}

public synchronized boolean TomarTenedorIzq() {
    return left.tomar();
}

public synchronized boolean TomarTenedorDer() {
    return right.tomar();
}

public synchronized void SoltarTenedores() {
    left.soltar();
    right.soltar();
}

public synchronized void Esperar() throws InterruptedException {
    estado = "esperando";
    System.out.println("Filósofo " + id + " está esperando...");
    wait(); // El filósofo espera hasta que los tenedores estén disponibles
}

```

```
}
```

```
public class Main {  
    public static void main(String[] args) {  
        // Crear los tenedores  
        Tenedor t1 = new Tenedor();  
        Tenedor t2 = new Tenedor();  
        Tenedor t3 = new Tenedor();  
        Tenedor t4 = new Tenedor();  
        Tenedor t5 = new Tenedor();  
  
        // Crear los filósofos  
        Filosofo f1 = new Filosofo(1, t1, t2);  
        Filosofo f2 = new Filosofo(2, t2, t3);  
        Filosofo f3 = new Filosofo(3, t3, t4);  
        Filosofo f4 = new Filosofo(4, t4, t5);  
        Filosofo f5 = new Filosofo(5, t5, t1);  
  
        // Iniciar los hilos de los filósofos  
        f1.start();  
        f2.start();  
        f3.start();  
        f4.start();  
        f5.start();  
    }  
}
```

Multiplicación de un vector por un escalar usando el enfoque de la Cena de los Filósofos

Se reutiliza la lógica de hilos concurrentes del problema de la Cena de los Filósofos, donde cada filósofo representa una unidad de trabajo. En este nuevo enfoque, cada "filósofo" es un hilo encargado de procesar una posición del vector y multiplicarla por un escalar. La sincronización que antes se daba para acceder a los tenedores, ahora se aplica sobre el acceso a la estructura compartida de resultados.

Código generalizado (con enfoque de filósofos):

```
main.java
1 class Tenedor {
2     private boolean disponible = true;
3
4     public synchronized boolean tomar() {
5         if (disponible) {
6             disponible = false;
7             return true;
8         }
9         return false;
10    }
11
12    public synchronized void soltar() {
13        disponible = true;
14    }
15 }
16
17 class FilosofoVector extends Thread {
18     private int id;
19     private int valor;
20     private int escalar;
21     private int[] resultado;
22     private Tenedor tenedor;
23
24     public FilosofoVector(int id, int valor, int escalar, int[] resultado, Tenedor tenedor) {
25         this.id = id;
26         this.valor = valor;
27         this.escalar = escalar;
28         this.resultado = resultado;
29         this.tenedor = tenedor;
30     }
31
32     @Override
33     public void run() {
34         try {
35             while (!tenedor.tomar()) {
36                 sleep(100); // espera activa, simulando bloqueo
37             }
38
39             System.out.println("Hilo " + id + " multiplicando " + valor + " x " + escalar);
40             int producto = valor * escalar;
41
42             synchronized (resultado) {
43                 resultado[id] = producto;
44             }
45
46             tenedor.soltar();
47
48         } catch (InterruptedException e) {
49             System.err.println("Error: " + e.getMessage());
50         }
51     }
52 }
```


Main con ejecución:

```
public class MainMultiplicacion {  
    public static void main(String[] args) {  
        int[] vector = {1, 2, 3, 4, 5};  
        int escalar = 3;  
        int[] resultado = new int[vector.length];  
  
        Tenedor[] tenedores = new Tenedor[vector.length];  
        for (int i = 0; i < tenedores.length; i++) {  
            tenedores[i] = new Tenedor();  
        }  
  
        FilosofoVector[] filosofos = new FilosofoVector[vector.length];  
        for (int i = 0; i < vector.length; i++) {  
            filosofos[i] = new FilosofoVector(i, vector[i], escalar, resultado, tenedores[i]);  
            filosofos[i].start();  
        }  
  
        // Esperamos a que terminen  
        for (FilosofoVector f : filosofos) {  
            try {  
                f.join();  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
  
        // Mostramos resultado  
        System.out.print("Resultado: ");  
        for (int val : resultado) {  
            System.out.print(val + " ");  
        }  
    }  
}
```

Resultado de ejecución

```
Hilo 0 multiplicando 1 x 3  
Hilo 4 multiplicando 5 x 3  
Hilo 3 multiplicando 4 x 3  
Hilo 1 multiplicando 2 x 3  
Hilo 2 multiplicando 3 x 3  
Resultado: 3 6 9 12 15  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Conclusiones

Romel Arenas Jiménez:

En conclusión, la experiencia de trabajar con mis compañeros de equipo para descargar y aprender a utilizar Apache NetBeans ha sido enriquecedora y valiosa. A lo largo de este proceso, hemos adquirido conocimientos fundamentales sobre el entorno de desarrollo integrado y su aplicación en la programación. La colaboración y el trabajo en equipo han sido esenciales para superar los desafíos y alcanzar nuestros objetivos. La experiencia adquirida en este proyecto nos será de gran utilidad en el transcurso de nuestras carreras, ya que nos ha permitido desarrollar habilidades prácticas y teóricas en la programación y el desarrollo de software. Además, hemos aprendido a valorar la importancia del trabajo en equipo y la comunicación efectiva para alcanzar metas comunes. Esta experiencia ha sido un paso importante en nuestro crecimiento profesional y estamos seguros de que nos beneficiará en el futuro. La utilización de Apache NetBeans nos ha abierto las puertas a nuevas posibilidades y oportunidades en el campo de la programación. Estamos agradecidos por esta experiencia y estamos ansiosos por aplicar nuestros conocimientos en futuros proyectos.

Jose Emiliano Jauregui Guzman:

Para esta actividad al principio fue complicado ya que para el sistema operativo ios no suele ser muy informativo al momento de las descargar estos programas y se tiene que investigar un poco mas sobre la instalación correcta. Al estar trabajando nuevamente con el problema de los filósofos que si no mal no recuerdo lo vimos en programación concurrente me resulto con mayor facilidad el entendimiento que tipo de problemas pueden presentar al momento de trabajar con el y aunque en esta ocasión fue un poco diferente la manera de trabajarlo, con ayuda de un equipo se logra su comprensión casi por completo y nos ayuda a experimentar las dudas que otros compañeros nos logran resolver y al ser tan fundamental este manejo de la herramientas para actividades futuras considero que se tiene que mejorar la dedicación que le pondremos a las actividades.

Angel Christian Lopez –

Conclusión sobre el Problema de la Cena de los Filósofos y la Multiplicación de Vectores:

El **Problema de la Cena de los Filósofos** es un clásico ejemplo en la programación concurrente que demuestra los desafíos de la sincronización en sistemas de múltiples hilos. En este caso, los filósofos representan hilos que intentan acceder de manera simultánea a recursos limitados (los tenedores), lo que genera potenciales problemas de **bloqueo y condiciones de carrera**. Este problema ilustra cómo es necesario gestionar el acceso a recursos compartidos para evitar que los hilos se bloqueen o produzcan resultados incorrectos.

Al aplicar este concepto al problema de **multiplicación de un vector por un escalar**, hemos utilizado el modelo de los filósofos para simular un entorno donde múltiples procesos (hilos) realizan una operación matemática (multiplicar un elemento del vector) sobre un recurso compartido (el resultado final). La sincronización se implementa mediante el uso de **hilos y semáforos** para evitar que los filósofos (hilos) accedan simultáneamente a los tenedores (recursos) sin coordinación, lo que puede causar inconsistencias en los resultados.

En este contexto, la **multiplicación de un vector por un escalar** se realiza en paralelo, lo que optimiza el tiempo de ejecución al distribuir las tareas entre varios hilos, asegurando que cada hilo realice la operación en un elemento del vector sin interferir con los demás. Sin embargo, la clave está en gestionar correctamente la sincronización entre los hilos, de modo que no haya bloqueos (deadlocks) ni condiciones de carrera, utilizando técnicas como **sincronización de métodos** y el uso de **bloqueos** adecuados para proteger el acceso a recursos compartidos.

En resumen, este ejercicio no solo aplica la teoría de la programación concurrente, sino que también muestra la importancia de **la sincronización** y el **manejo de recursos compartidos** en sistemas de múltiples hilos. La correcta implementación de estos conceptos es esencial para evitar problemas de bloqueos, inconsistencias en los resultados y otros errores comunes en sistemas concurrentes.

Adair Espinosa Estrada

Durante esta primera etapa del proyecto integrador, logramos sentar las bases para comprender y aplicar los conceptos fundamentales de la concurrencia en sistemas operativos, utilizando como referencia el clásico problema de la cena de los filósofos comelones con una correcta gestión de hilos y sincronización mediante bloques `synchronized`, comprendimos los desafíos que surgen cuando múltiples procesos (o hilos) compiten por recursos compartidos.

También la implementación de la clase `tenedor`, que permitió modelar el recurso compartido con control de acceso concurrente, previniendo condiciones de carrera, asimismo, se logró evitar el bloqueo mutuo al diseñar correctamente las acciones de toma y liberación de recursos, y al incorporar tiempos de espera y estados bien definidos para cada filósofo. Cabe destacar que este problema ya lo había abordado previamente en otra materia, utilizando C++ dentro del entorno de visual estudio, sin embargo, esta nueva implementación me permitió comparar enfoques y profundizar en cómo distintos lenguajes gestionan la concurrencia y la sincronización.

Angel Alejandro Alcala Aguilar

Para concluir, recuerdo haber realizado anteriormente una actividad sobre “los filósofos comelones” al menos con ese nombre la conocí en ese entonces, recuerdo que había múltiples formas de resolverlo, al inicio nos plantearon el problema desde un punto de vista más psicológico, para que contempláramos distintos escenarios, había la posibilidad de que alguno de los filósofos hubiera comido antes de sentarse por lo que no tendría mucha hambre, mientras que otro podría estar muy hambriento, aplicado a la programación sería asignándoles ordenes de prioridad a cada proceso, es una opción muy buena pues destina los recursos siempre al proceso que tenga una mayor prioridad, aunque tiene el inconveniente de que los procesos con menos prioridad nunca consigan recursos o tarden mucho para ello. Igualmente se pueden solucionar aplicando distintos métodos o propiamente haciendo uso de nuestros procesadores de múltiple núcleo

Realmente esta ha sido una buena actividad, se me refresco la memoria sobre este problema y es un claro indicativo de como debemos ser como ingenieros, siempre buscando soluciones ingeniosas a los problemas que se presenten.

Referencias

- Bellido Quintero, E. (2013). *Instalación y actualización de sistemas operativos UF0852*. Recuperado de <https://www.iceditorial.mx/e-books/6253-instalacion-y-actualizacion-de-sistemas-operativos-ifct0309--9788483647783.html>
- Blog Bitix (2018). *El problema de concurrencia de la cena de los filósofos resuelto con Java*. Recuperado de <https://picodotdev.github.io/blog-bitix/2018/02/el-problema-de-concurrencia-de-la-cena-de-los-filosofos-resuelto-con-java/>
- McIver McHoes, A. y M. Flynn, I. (2011). *Sistemas Operativos*. Recuperado de https://www.academia.edu/18177488/191183465_Sistemas_Operativos_Flynn_Mchoes
- Nti [nombre de usuario]. (2016). *Java la cena de los filósofos*. Recuperado de: <https://www.lawebdelprogramador.com/foros/Java/1531083-Cena-de-filosofos.html>

**UVM****Universidad
del Valle de México**
LAUREATE INTERNATIONAL UNIVERSITIES*

MATERIA

Sistemas Operativos

ACTIVIDAD 7: Proyecto Integrador Etapa 2

UNIDAD 4: Archivos

EQUIPO: Windows 4

Ángel Alejandro Alcalá Aguilar

Romel Arenas Jiménez

Ángel Christian López

Adair Espinosa Estrada

José Emiliano Jauregui Guzmán

NOMBRE DEL DOCENTE: Manuel Triana Vega

FECHA: lunes 28 de abril de 2025

Indicé de contenido**Etapas del Proyecto integrador**

Introducción.....	17
II. Interbloques	
2.1 Aplicación de los principios de interbloqueo.....	18
2.2 Resultados	31
Conclusiones.....	32
Referencias.....	34

Introducción

En este proyecto integrador, aplicaré los conocimientos adquiridos a lo largo del curso para desarrollar algoritmos que manejen concurrencia y elementos de seguridad en un sistema operativo de código abierto como Linux. A través de la revisión de los materiales sugeridos y las actividades realizadas, desarrollaré los apartados indicados para cada etapa del proyecto. En particular, me enfocaré en la aplicación de los principios de interbloqueo para garantizar la eficiencia y la seguridad en el manejo de recursos compartidos. El objetivo es demostrar mi comprensión de las bases de la programación y su aplicación en un entorno real. Con este proyecto, busco fortalecer mis competencias y lograr el fin de formación planteado. A continuación, presento los resultados de mi investigación y desarrollo.

II. Interbloqueos

2.1 Aplicación de los principios de interbloqueo

Desarrolla los ejercicios en los que apliques los principios del interbloqueo. Para ello realiza lo siguiente:

- a) Realiza un programa que maneje una lista de contactos de agenda que incluya: nombre, e-mail y teléfono.
- b) Revisa los ejemplos de interbloqueo 1/2 y 2/2 (Diapositivas - Páginas 31 y 32) que aparecen en el documento Capítulo 7. Interbloqueos.

Ejemplo de interbloqueo (1/2)

// Versión con posible interbloqueo

```
struct nodo {
    struct nodo *siguiente;
    /* otros campos */
};

struct lista {
    pthread_mutex_t mutex_lista;
    struct nodo *primer_nodo;
};

void mover_elemento_de_lista(struct lista *origen, struct lista *destino,
    struct nodo *elemento, int posicion_destino) {
    pthread_mutex_lock(&origen->mutex_lista);
    pthread_mutex_lock(&destino->mutex_lista);
```

```

/* mueve el elemento a la lista destino */
pthread_mutex_unlock(&origen->mutex_lista);
pthread_mutex_unlock(&destino->mutex_lista);
}

```

Ejemplo de interbloqueo (2/2)

// Versión libre de interbloqueos

```

void mover_elemento_de_lista(struct lista *origen, struct lista* destino,
    struct nodo *elemento, int posicion_destino) {
    if (origen < destino) {
        pthread_mutex_lock(&origen->mutex_lista);
        pthread_mutex_lock(&destino->mutex_lista);
    }
    else {
        pthread_mutex_lock(&destino->mutex_lista);
        pthread_mutex_lock(&origen->mutex_lista);
    }

    /* mueve el elemento a la lista destino */

    pthread_mutex_unlock(&origen->mutex_lista);
    pthread_mutex_unlock(&destino->mutex_lista);
}

```

- c) Con base en el ejemplo anterior realiza la modificación de la lista de contactos, evitando los bloqueos que supone la primera diapositiva.

```

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

typedef struct contacto {
    char nombre[100];
    char email[100];
    char telefono[20];
    struct contacto *siguiente;
}

```

```

} contacto;

typedef struct lista {
    pthread_mutex_t mutex_lista;
    contacto *primer_contacto;
} lista;

void imprimir_lista(const char *titulo, lista *l) {
    pthread_mutex_lock(&l->mutex_lista);
    printf("%s:\n", titulo);
    contacto *c = l->primer_contacto;
    while (c) {
        printf(" - %s | %s | %s\n", c->nombre, c->email, c->telefono);
        c = c->siguiente;
    }
    pthread_mutex_unlock(&l->mutex_lista);
    printf("\n");
}

//Contacto al inicio
void insertar_contacto(lista *l, contacto *nuevo) {
    pthread_mutex_lock(&l->mutex_lista);
    nuevo->siguiente = l->primer_contacto;
    l->primer_contacto = nuevo;
    pthread_mutex_unlock(&l->mutex_lista);
}

//Movimineto de lista

void mover_contacto_entre_listas(lista *origen, lista *destino,
                                contacto *c, int posicion_destino) {
    if (origen < destino) {
        pthread_mutex_lock(&origen->mutex_lista);
        pthread_mutex_lock(&destino->mutex_lista);
    } else {
        pthread_mutex_lock(&destino->mutex_lista);
        pthread_mutex_lock(&origen->mutex_lista);
    }
}

```

```
// Quitar de origen
contacto **curr = &origen->primer_contacto;
while (*curr && *curr != c) {
    curr = &(*curr)->siguiente;
}
if (*curr == c) {
    *curr = c->siguiente;

    // Insertar en destino
    contacto **pos = &destino->primer_contacto;
    for (int i = 0; i < posicion_destino && *pos; i++) {
        pos = &(*pos)->siguiente;
    }
    c->siguiente = *pos;
    *pos = c;
}

pthread_mutex_unlock(&origen->mutex_lista);
pthread_mutex_unlock(&destino->mutex_lista);
}

int main() {
    // Inicializar listas
    lista lista_1 = { PTHREAD_MUTEX_INITIALIZER, NULL };
    lista lista_2 = { PTHREAD_MUTEX_INITIALIZER, NULL };

    // Crear contactos
    contacto *jose = malloc(sizeof(contacto));
    strcpy(jose->nombre, "Jose Jauregui");
    strcpy(jose->email, "jose@gmail.com");
    strcpy(jose->telefono, "4426256789");

    contacto *romel = malloc(sizeof(contacto));
    strcpy(romel->nombre, "Romel Arenas");
    strcpy(romel->email, "romel@gmail.com");
    strcpy(romel->telefono, "3398927792");
```

```
// Insertar en lista 1
insertar_contacto(&lista_1, jose);
insertar_contacto(&lista_1, romel);

// Mostrar listas antes
printf("Antes de mover contacto:\n");
imprimir_lista("Lista 1", &lista_1);
imprimir_lista("Lista 2", &lista_2);

// Mover contacto (Ana) a lista 2 en posición 0
mover_contacto_entre_listas(&lista_1, &lista_2, romel, 0);

// Mostrar listas después
printf("Después de mover contacto:\n");
imprimir_lista("Lista 1", &lista_1);
imprimir_lista("Lista 2", &lista_2);

// Liberar memoria
free(jose);
free(romel);

return 0;
}
```

Ejecución

```

main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <pthread.h>
5
6  typedef struct contacto {
7      char nombre[100];
8      char email[100];
9      char telefono[20];
10     struct contacto *siguiente;
11 } contacto;
12
13 typedef struct lista {
14     pthread_mutex_t mutex_lista;
15     contacto *primer_contacto;
16 } lista;
17
18 void imprimir_lista(const char *titulo, lista *l) {
19     pthread_mutex_lock(&l->mutex_lista);
20     printf("%s:\n", titulo);
21     contacto *c = l->primer_contacto;
22     while (c) {
23         printf("  - %s | %s | %s\n", c->nombre, c->email, c->telefono);
24         c = c->siguiente;
25     }
26     pthread_mutex_unlock(&l->mutex_lista);
27     printf("\n");
28 }
29
30 //Contacto al inicio
31 void insertar_contacto(lista *l, contacto *nuevo) {
32     pthread_mutex_lock(&l->mutex_lista);
33     nuevo->siguiente = l->primer_contacto;
34     l->primer_contacto = nuevo;
35     pthread_mutex_unlock(&l->mutex_lista);
36 }
37
38 //Movimineto de lista
39
40 void mover_contacto_entre_listas(lista *origen, lista *destino,
41     contacto *c, int posicion_destino) {
42     if (origen < destino) {
43         pthread_mutex_lock(&origen->mutex_lista);
44         pthread_mutex_lock(&destino->mutex_lista);
45     } else {
46         pthread_mutex_lock(&destino->mutex_lista);
47         pthread_mutex_lock(&origen->mutex_lista);
48     }
49
50     // Quitar de origen
51     contacto **curr = &origen->primer_contacto;
52     while (*curr && *curr != c) {
53         curr = &(*curr)->siguiente;
54     }
55     if (*curr == c) {
56         *curr = c->siguiente;
57     }
58 }

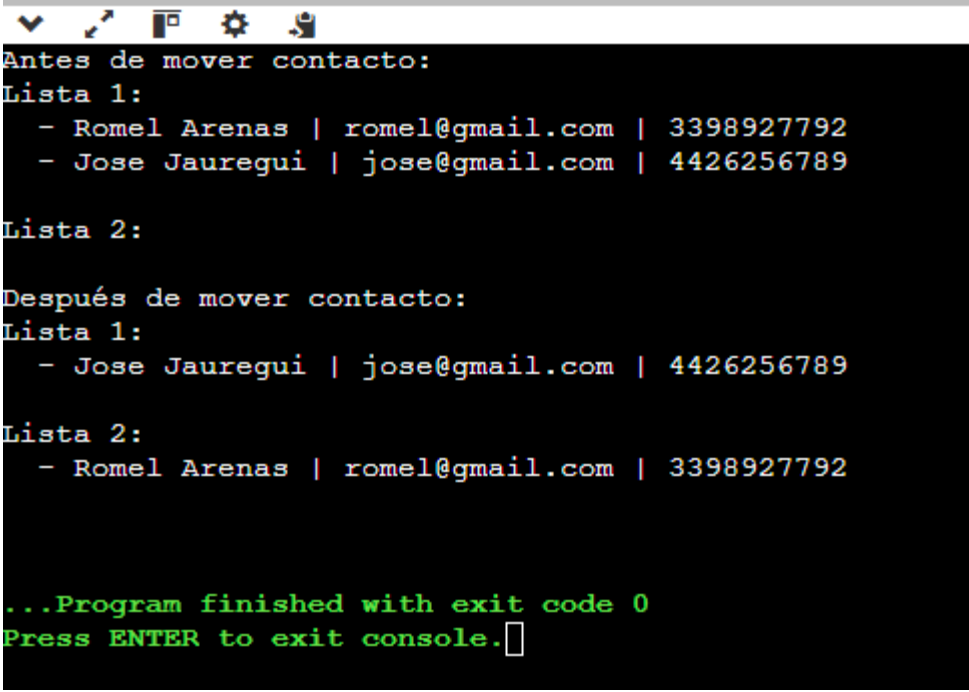
```

```

58     // Insertar en destino
59     contacto **pos = &destino->primer_contacto;
60     for (int i = 0; i < posicion_destino && *pos; i++) {
61         pos = &(*pos)->siguiente;
62     }
63     c->siguiente = *pos;
64     *pos = c;
65 }
66
67 pthread_mutex_unlock(&origen->mutex_lista);
68 pthread_mutex_unlock(&destino->mutex_lista);
69 }
70
71 int main() {
72     // Inicializar listas
73     lista lista_1 = { PTHREAD_MUTEX_INITIALIZER, NULL };
74     lista lista_2 = { PTHREAD_MUTEX_INITIALIZER, NULL };
75
76     // Crear contactos
77     contacto *jose = malloc(sizeof(contacto));
78     strcpy(jose->nombre, "Jose Jauregui");
79     strcpy(jose->email, "jose@gmail.com");
80     strcpy(jose->telefono, "4426256789");
81
82     contacto *romel = malloc(sizeof(contacto));
83     strcpy(romel->nombre, "Romel Arenas");
84     strcpy(romel->email, "romel@gmail.com");
85     strcpy(romel->telefono, "3398927792");
86
87     // Insertar en lista 1
88     insertar_contacto(&lista_1, jose);
89     insertar_contacto(&lista_1, romel);
90
91     // Mostrar listas antes
92     printf("Antes de mover contacto:\n");
93     imprimir_lista("Lista 1", &lista_1);
94     imprimir_lista("Lista 2", &lista_2);
95
96     // Mover contacto (Ana) a lista 2 en posición 0
97     mover_contacto_entre_listas(&lista_1, &lista_2, romel, 0);
98
99     // Mostrar listas después
100    printf("Después de mover contacto:\n");
101    imprimir_lista("Lista 1", &lista_1);
102    imprimir_lista("Lista 2", &lista_2);
103
104    // Liberar memoria
105    free(jose);
106    free(romel);
107
108    return 0;
109 }

```

Output de código



```

Antes de mover contacto:
Lista 1:
- Romel Arenas | romel@gmail.com | 3398927792
- Jose Jauregui | jose@gmail.com | 4426256789

Lista 2:

Después de mover contacto:
Lista 1:
- Jose Jauregui | jose@gmail.com | 4426256789

Lista 2:
- Romel Arenas | romel@gmail.com | 3398927792

...Program finished with exit code 0
Press ENTER to exit console.

```

- d) Modifica el código adjunto para realizar esta tarea a través del patrón de diseño singleton y además el uso de threads en java.

El patrón singleton se implementó en la clase

```

class ListaSegura {
    private static ListaSegura instance;
    private final Lock lock = new ReentrantLock();
    private Contacto primerContacto;

    private ListaSegura() {
        primerContacto = null;
    }

    public static ListaSegura getInstance() {
        if (instance == null) {
            instance = new ListaSegura();
        }
        return instance;
    }
}

```

El objetivo es garantizar que sin importar cuántos hilos u objetos accedan a la lista, se mantenga una única instancia de la clase lista segura, esto siguiendo el patrón de diseño Singleton.

En cuanto al uso de threads se están utilizando para insertar y mover contactos de manera concurrente en la lista segura.

Los threads permiten ejecutar tareas en paralelo, lo que es útil en aplicaciones que deben realizar múltiples operaciones de manera simultánea sin bloquear el flujo principal.

- Estos ejecutan un bloque de código en paralelo

```
Thread t1 = new Thread() -> {
    listaSegura.insertarContacto(jose);
};

Thread t2 = new Thread() -> {
    listaSegura.insertarContacto(romel);
};
```

En los métodos_insertar contacto y mover contacto de lista segura están siendo ejecutados por diferentes hilos para permitir la inserción y movimiento concurrente de los contactos en la lista.

Los hilos se inician

```
t1.start();
t2.start();
```

Esto pone en ejecución el código definido en la clase thread

La sincronización de los hilos se hace con lock esto garantiza que la lista no se vea afectada por condiciones de carrera garantizando que solo un hilo pueda acceder a las operaciones críticas en un momento dado.

Insertar contacto y mover contacto están protegidos por un bloque `lock lock` al principio y `lock.unlock` al final.

Asegurando que solo un hilo pueda realizar estas operaciones a la vez.

```
public void insertarContacto(Contacto contacto) {
    lock.lock();
    try {
        contacto.siguiente = primerContacto;
        primerContacto = contacto;
    } finally {
        lock.unlock();
    }
}
```

Se usa el método join para esperar que cada hilo termine su ejecución antes de proceder con la siguiente operación

```
t1.join();
t2.join();
```

Esto permite bloquear el hilo principal hasta que el hilo correspondiente haya terminado de ejecutarse.

Captura de pantalla del Código en java

```

2  import java.util.concurrent.locks.Lock;
3  import java.util.concurrent.locks.ReentrantLock;
4  public class Proyecto2 {
5      public static void main(String[] args) {
6          ListaSegura listaSegura = ListaSegura.getInstance();
7
8          Contacto jose = new Contacto("Jose Jauregui", "jose@gmail.com", "4426256789");
9          Contacto romel = new Contacto("Romel Arenas", "romel@gmail.com", "3398927792");
10
11         Thread t1 = new Thread() -> {
12             listaSegura.insertarContacto(jose);
13         };
14
15         Thread t2 = new Thread() -> {
16             listaSegura.insertarContacto(romel);
17         };
18         t1.start();
19         t2.start();
20         try {
21             t1.join();
22             t2.join();
23         } catch (InterruptedException e) {
24             e.printStackTrace();
25         }
26         listaSegura.imprimirLista();
27
28         Thread t3 = new Thread() -> {
29             listaSegura.moverContacto(jose, 0);
30         };
31
32         Thread t4 = new Thread() -> {
33             listaSegura.moverContacto(romel, 1);
34         };
35         t3.start();
36         t4.start();
37         try {
38             t3.join();
39             t4.join();
40         } catch (InterruptedException e) {
41             e.printStackTrace();
42         }
43         listaSegura.imprimirLista();
44     }
45 }
46
47 class ListaSegura {
48     private static ListaSegura instance;
49     private final Lock lock = new ReentrantLock();
50     private Contacto primerContacto;
51 
```

```

76     try {
77         Contacto current = primerContacto;
78         Contacto prev = null;
79
80         while (current != null) {
81             if (current == contacto) {
82                 if (prev != null) {
83                     prev.siguiente = current.siguiente;
84                 } else {
85                     primerContacto = current.siguiente;
86                 }
87                 break;
88             }
89             prev = current;
90             current = current.siguiente;
91         }
92         current.siguiente = primerContacto;
93         primerContacto = current;
94     } finally {
95         lock.unlock();
102         while (current != null) {
103             System.out.println(current);
104             current = current.siguiente;
105         }
106     } finally {
107         lock.unlock();
108     }
109 }
110
111 class Contacto {
112     String nombre;
113     String email;
114     String telefono;
115     Contacto siguiente;
116     public Contacto(String nombre, String email, String telefono) {
117         this.nombre = nombre;
118         this.email = email;
119         this.telefono = telefono;
120         this.siguiente = null;
121     }
122     @Override
123     public String toString() {
124         return "Nombre: " + nombre + ", Email: " + email + ", Telefono: " + telefono;
125     }
126 }

```

Resultado

Output - Run (proyecto2) x

```

--- exec:3.1.0:exec (default-cli) @ proyecto2 ---
Nombre: Jose Jauregui, Email: jose@gmail.com, Telefono: 4426256789
Nombre: Romel Arenas, Email: romel@gmail.com, Telefono: 3398927792
Nombre: Romel Arenas, Email: romel@gmail.com, Telefono: 3398927792
Nombre: Jose Jauregui, Email: jose@gmail.com, Telefono: 4426256789
-----

```

- e) Escribe un programa de tal forma que se resuelva el problema de los interbloqueos utilizando algún recurso para poder generar el método denominado `usar_ambos_recurso()`. Puede ser una estructura, una matriz, un arreglo, etc.

Descripción del programa

```
Recurso[] recursos = new Recurso[2];
recursos[0] = new Recurso("Recurso A");
recursos[1] = new Recurso("Recurso B");
```

Este arreglo contiene los dos recursos compartidos recurso A y recurso B permitiendo compartir los recursos entre hilos.

Clase (Recurso)

- f) Representa un recurso que puede ser usado por un hilo
- g) Tiene el método `usar` que imprime que hilo lo está utilizando

Clase (contacto)

Es una estructura auxiliar que almacena información personal como nombre, correo y teléfono.

Ilustrar una acción crítica (uso del recurso) con salida visible en consola.

Clase (Tarea implements Runnable)

Tarea que ejecuta cada hilo.

Cada hilo tiene un índice de inicio para simular que puede intentar acceder a los recursos en distinto orden.

Método `usar_ambos_recurso ()`

Es para evitar interbloqueo usando lo siguiente

```
public void usar_ambos_recurso() {
    int primero = Math.min(indiceInicio, (indiceInicio + 1) % 2);
    int segundo = Math.max(indiceInicio, (indiceInicio + 1) % 2);
```

Esto permite bloquear los recursos en el mismo orden, sin importar el orden de acceso original del hilo, lo que rompe las causas del interbloqueo.

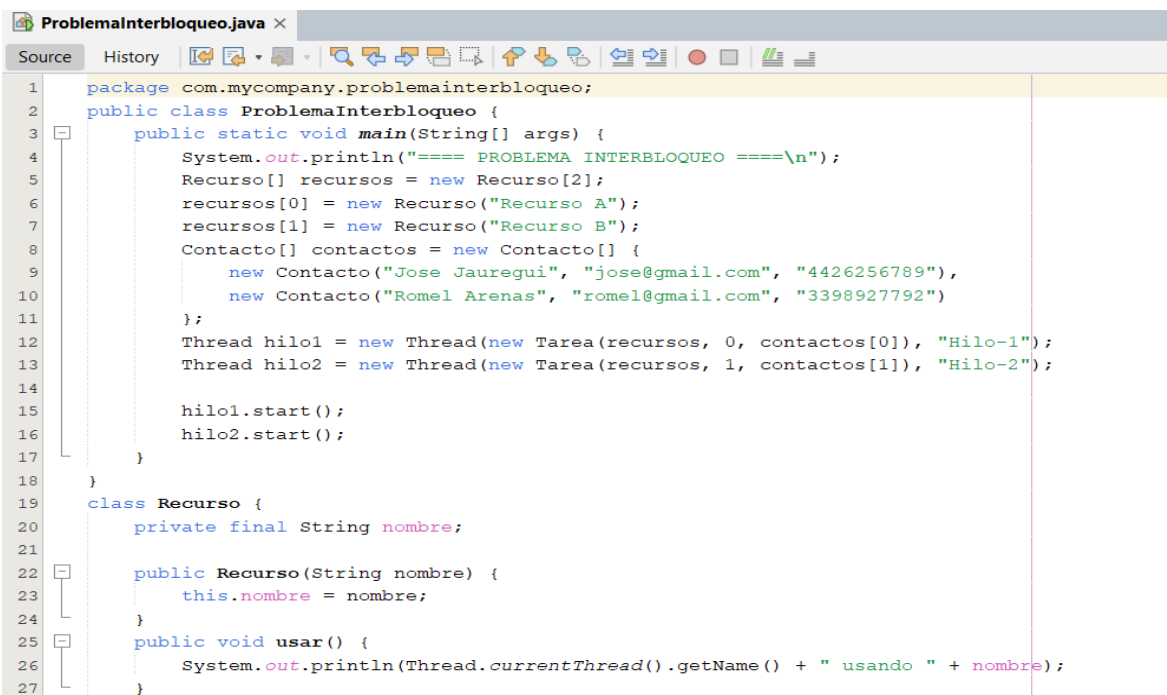
Se sincroniza

```
synchronized (recursos[primero]) {
    recursos[primero].usar();
    try { Thread.sleep(50); } catch (InterruptedException e) { }

    synchronized (recursos[segundo]) {
        recursos[segundo].usar();
        contacto.imprimirContacto();
        contacto.imprimirContacto();
    }
}
```

Garantizando que no se produzca interbloqueo ya que todos los hilos adquieren los recursos en un orden.

Captura de pantalla del Código en java



```
ProblemaInterbloqueo.java x
Source History
1 package com.mycompany.problemainterbloqueo;
2 public class ProblemaInterbloqueo {
3     public static void main(String[] args) {
4         System.out.println("==== PROBLEMA INTERBLOQUEO ====\n");
5         Recurso[] recursos = new Recurso[2];
6         recursos[0] = new Recurso("Recurso A");
7         recursos[1] = new Recurso("Recurso B");
8         Contacto[] contactos = new Contacto[] {
9             new Contacto("Jose Jauregui", "jose@gmail.com", "4426256789"),
10            new Contacto("Romel Arenas", "romel@gmail.com", "3398927792")
11        };
12        Thread hilo1 = new Thread(new Tarea(recursos, 0, contactos[0]), "Hilo-1");
13        Thread hilo2 = new Thread(new Tarea(recursos, 1, contactos[1]), "Hilo-2");
14
15        hilo1.start();
16        hilo2.start();
17    }
18 }
19 class Recurso {
20     private final String nombre;
21
22     public Recurso(String nombre) {
23         this.nombre = nombre;
24     }
25     public void usar() {
26         System.out.println(Thread.currentThread().getName() + " usando " + nombre);
27     }
28 }
```

```

28 }
29 class Contacto {
30     String nombre;
31     String email;
32     String telefono;
33     public Contacto(String nombre, String email, String telefono) {
34         this.nombre = nombre;
35         this.email = email;
36         this.telefono = telefono;
37     }
38
39     public void imprimirContacto() {
40         System.out.println("Nombre: " + nombre + ", Email: " + email + ", Telefono: " + telefono);
41     }
42 }
43 class Tarea implements Runnable {
44     private final Recurso[] recursos;
45     private final int indiceInicio;
46     private final Contacto contacto;
47     public Tarea(Recurso[] recursos, int indiceInicio, Contacto contacto) {
48         this.recursos = recursos;
49         this.indiceInicio = indiceInicio;
50         this.contacto = contacto;
51     }
52     @Override
53     public void run() {
54         usar_ambos_recursos();
55     }
56     public void usar_ambos_recursos() {
57         int primero = Math.min(indiceInicio, (indiceInicio + 1) % 2);
58         int segundo = Math.max(indiceInicio, (indiceInicio + 1) % 2);
59
60         synchronized (recursos[primero]) {
61             recursos[primero].usar();
62             try { Thread.sleep(50); } catch (InterruptedException e) { }
63
64             synchronized (recursos[segundo]) {
65                 recursos[segundo].usar();
66                 contacto.imprimirContacto();
67                 contacto.imprimirContacto();
68             }
69         }
70     }
71 }

```

Resultado

Output - Run (problemaInterbloqueo) X

```

==== PROBLEMA INTERBLOQUEO ====

Hilo-1 usando Recurso A
Hilo-1 usando Recurso B
Nombre: Jose Jauregui, Email: jose@gmail.com, Telefono: 4426256789
Nombre: Jose Jauregui, Email: jose@gmail.com, Telefono: 4426256789
Hilo-2 usando Recurso A
Hilo-2 usando Recurso B
Nombre: Romel Arenas, Email: romel@gmail.com, Telefono: 3398927792
Nombre: Romel Arenas, Email: romel@gmail.com, Telefono: 3398927792

```

2.2 Resultados

Reflexión sobre las mejores prácticas para evitar interbloqueos

Los interbloqueos son un problema común en los sistemas operativos que pueden causar graves consecuencias en la eficiencia y la estabilidad del sistema. Para evitar o resolver estos problemas, es fundamental implementar mejores prácticas en el diseño y la gestión de los procesos y recursos del sistema.

Mejores prácticas para evitar interbloqueos

- Evitar la exclusión mutua innecesaria: limitar el uso de exclusión mutua solo a los recursos que realmente la requieren.
- Implementar un orden de adquisición de recursos: establecer un orden específico para la adquisición de recursos para evitar ciclos de espera.
- Utilizar timeouts: implementar timeouts para detectar y resolver interbloqueos de manera oportuna.
- Diseñar algoritmos de detección de interbloqueos: implementar algoritmos que detecten y resuelvan interbloqueos de manera automática.

Solución de interbloqueos

Para solucionar o resolver interbloqueos, es importante:

- Identificar la causa raíz del problema: analizar el sistema y los procesos para determinar la causa del interbloqueo.
- Liberar recursos: liberar recursos bloqueados para permitir que otros procesos continúen ejecutándose.
- Reconfigurar el sistema: reconfigurar el sistema para evitar que se produzcan interbloqueos similares en el futuro.

En resumen, la prevención y la detección temprana de interbloqueos son fundamentales para garantizar la eficiencia y la estabilidad de un sistema operativo. Al implementar mejores prácticas en el diseño y la gestión de procesos y recursos, podemos minimizar el riesgo de interbloqueos y asegurar un funcionamiento óptimo del sistema.

Conclusiones

Romel Arenas Jiménez:

En conclusión, este proyecto ha sido una experiencia transformadora que me ha permitido adquirir conocimientos valiosos sobre la gestión de procesos y recursos en sistemas operativos. He aprendido a identificar y resolver problemas de interbloqueos, y a implementar mejores prácticas para evitarlos. La colaboración y el aprendizaje compartido en equipo han sido fundamentales en este proceso, y he podido apreciar la importancia de trabajar en equipo y aprender de los demás. Cada miembro ha aportado su perspectiva y conocimientos únicos, lo que ha enriquecido mi comprensión del tema. Estoy seguro de que estos conocimientos y habilidades serán de gran utilidad en mi futuro profesional, y me siento preparado para enfrentar nuevos desafíos y aplicar mis conocimientos en situaciones reales. La experiencia adquirida me ha enseñado a ser más crítico y analítico en mi enfoque hacia los problemas, y a valorar la importancia de la planificación y la organización. En general, este proyecto ha sido una experiencia enriquecedora que me ha permitido crecer como profesional y como persona. Estoy emocionado de aplicar lo que he aprendido en mi carrera y seguir creciendo como profesional. Con esta experiencia, me siento más confiado en mi capacidad para abordar desafíos complejos y encontrar soluciones efectivas.

Jose Emiliano Jauregui Guzman:

El desarrollo de este trabajo me permitió comprender en profundidad los conceptos teóricos y prácticos relacionados con los interbloqueos en sistemas operativos, así como las estrategias para evitarlos y resolverlos de forma eficiente. A través de la implementación de una agenda de contactos y la aplicación del patrón Singleton junto con el uso de hilos en Java o en C en muchos casos, pude experimentar de primera mano cómo pueden ocurrir situaciones de interbloqueo y, lo más importante, cómo prevenirlas mediante una correcta gestión de los recursos compartidos. Me siento satisfecho con el aprendizaje obtenido, ya que logré aplicar los principios vistos en clase a casos prácticos y reales, fortaleciendo así mis habilidades en programación concurrente y diseño de software seguro y eficiente. Esta experiencia me ha brindado herramientas clave que podré aplicar en futuros proyectos, tanto académicos como profesionales. Dando paso a la última etapa del proyecto para culminar la materia.

Referencias

- Álvarez, C. (2014). *Ejemplo de Java Singleton (Patrones y ClassLoaders)*
Recuperado de <https://www.arquitecturajava.com/ejemplo-de-java-singleton-patrones-classloaders/>
- Apasoft Training (Productor). (21 de mayo de 2019). *Linux desde Cero. Sistemas de Ficheros*. Recuperado de https://www.youtube.com/watch?v=0NUorN_uadI
- Carreto, J., García, F., Miguel de, P. y Pérez, F. (2007). *Sistemas operativos: Una visión aplicada* de http://laurel.datsi.fi.upm.es/_media/docencia/asignaturas/dso/interbloqueos-4pp.pdf
- Giordana, D. (Productor). (12 de Diciembre de 2017). *El sistema de archivos en sistemas Linux*. Recuperado de <https://www.youtube.com/watch?v=MEb31TzuAVU>