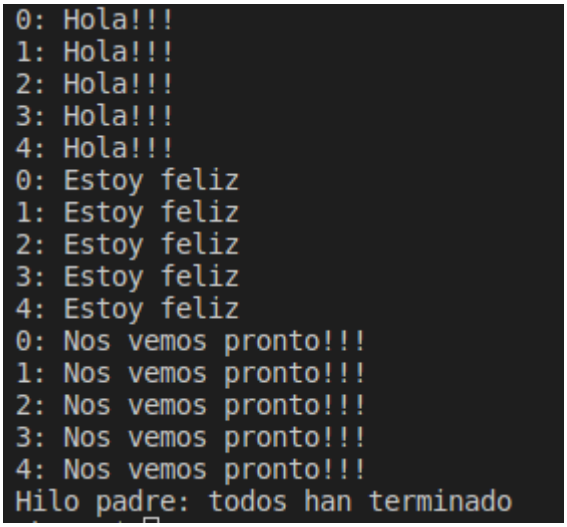
Programación Concurrente, Marzo 2023

Práctica 3

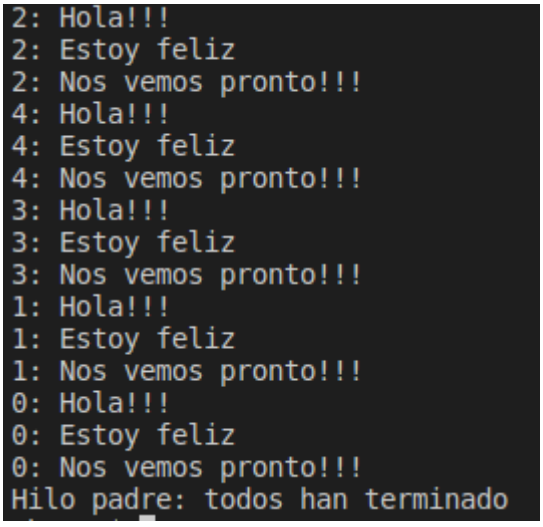
Martínez Buenrostro Jorge Rafael.
Universidad Autónoma Metropolitana
Unidad Iztapalapa, México
*molap*96*@gmail.com*

Resultado Ejercicio B

En las siguientes figuras podemos ver lo que muestra en la terminal el programa original y el programa con el uso de mutex.



Salida del programa original



Salida del programa con uso de mutex

Por las imágenes anteriores podría parecer que el comportamiento del programa cambió pero el uso de mutex se encuentra en el mismo lugar que el programa original. Este comportamiento se debe a que cuando desbloquea el candado el mismo hilo lo vuelve a tomar inmediatamente; cada ejecución del programa es distinta ya que el orden en el que cada hilo se ejecuta varía.

Resultado Ejercicio D

En la siguiente imágen podemos ver las distintas versiones de sincronización que nos da el programa *frecuencias.c*. Para poder analizar cada una se quitan los comentarios en pares iguales para poder ver el comportamiento.

```
//V1: el derecho de acceso se guarda toda la ejecución
//pthread_mutex_lock(&mi_candado);

for(num = 1; num<=10; num++){
    //V2: el derecho de acceso se guarda una iteración
    //pthread_mutex_lock(&mi_candado);
            cont = 0;
            for (j= (mis_limites->ini); j<= (mis_limites->fin); j++)
                if (nums[j] == num)
                    cont++;
            //V3.el derecho de acceso se guarda solo durante la escritura
            //pthread_mutex_lock(&mi_candado);
            frec[num-1] += cont;
            //V3.Se libera el derecho de acceso
            //pthread_mutex_unlock(&mi_candado);
    //V2: Se libera el derecho de acceso
    //pthread_mutex_unlock(&mi_candado);
}
// V1. Se libera el derecho de acceso
//pthread_mutex_unlock(&mi_candado);
```

Versiones de sincronización del programa *frecuencias.c*

- **¿Qué repercusiones tienen para la coherencia de los datos cada una de ellas?**

  Para el caso en el que todas las versiones de sincronizacioń se encuentran comentadas la repercución radica en que ambos hilos están escribiendo sobre el mismo arreglo sus resultados, por lo que se puede dar el caso que la inserción de datos de un hilo afecte los resultados ya puestos por el otro hilo.

  Para el caso de la versión de sincronización **V1** la coherencia de los datos se mantiene íntegra ya que el acceso al arreglo de frecuencias lo mantiene cada hilo durante toda su ejecución.

  Para el caso de la versión de sincronización **V2 l**a coherencia de los datos se mantiene íntegra ya que el acceso al arreglo de frecuencias lo mantiene cada hilo durante cada iteración.

  Para el caso de la versión de sincronización **V3 l**a coherencia de los datos se mantiene íntegra ya que el acceso al arreglo de frecuencias lo mantiene cada hilo solamente durante la escritura sobre el arreglo de frecuencias.

- **¿Qué repercusiones tienen para la eficiencia del programa,suponga que N es muy grande?**

  Si el programa no tuviera algún tipo de sincronización al **N** ser muy grande corremos el riesgo de que cada hilo sobreescriba o afecte los resultados escritos del otro hilo. En el caso de la **V1** no se aprovecharán las ventajas de hilos ya que se correrían de manera secuencial. En el caso de la **V2** el acceso lo tendría cada hilo por mucho tiempo ya que lo guarda sobre cada iteración. Mientras

que en el caso de **V3** no importa que tan grande es **N** ya que el acceso solo tiene el hilo al momento de escribir un resultado en el arreglo de frecuencias.

- **¿Con cuál se quedaría usted?**
  Para poder usar al máximo el potencial de los hilos personalmente usaría la versión 3 de sincronización ya que el acceso lo guarda cada hilo al momento de la escritura de su resultado.

Resultado Inciso E

Modifique el programa *frecuencias.c* para que se tenga un mutex por cada celda del arreglo de frecuencias; en la siguiente figura se muestran las modificaciones hechas al programa.

- La primera modificación es crear un arreglo de candados del mismo tamaño que los otros arreglos.

```
// Candados con los parámetros por defecto (abierto)
pthread_mutex_t mis_candados[TAM];
```

- La segunda modificación es inicializar cada candado del arreglo.

```
for(i=0;i<TAM;i++)
    pthread_mutex_init(&mis_candados[i], NULL);
```

- La tercera modificación se hace dentro de la función **Frecuencia;** en la que abrimos el candado que corresponde al número que se está evaluando, después de que el hilo escribe en el arreglo de frecuencias se cierra el candado.

```
pthread_mutex_lock(&mis_candados[num-1]);
frec[num-1] += cont;
pthread_mutex_unlock(&mis_candados[num-1]);
```

- La cuarta modificación consiste en destruir cada candado del arreglo

```
// Se destruyen los candados
for(i=0;i<TAM;i++)
    pthread_mutex_destroy(&mis_candados[i]);
```

¿Cuáles son los pros y los contras de hacer esto?

Los pros serían que cada índice del arreglo de resultados puede ser usado por un candado único, en clase hemos platicado de la analogía de los casilleros, entonces cada índice del arreglo ahora es un casillero que tiene un candado único. Un contra es que si no se manejan de manera adecuada podrían quedar candados sin destruir, o incluso candados sin cerrar.

Resultado Inciso F

Agregue al programa *frecuencias.c* un hilo que determine cuál fue la frecuencia más alta de los números en el arreglo. El hilo deja el resultado de su trabajo en una variable compartida que el hilo padre leerá para desplegar el o los números que tuvieron esa frecuencia.

Enlace GDB online

- https://onlinegdb.com/1fnhTWuRX

Código *frecuencia.c*

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#define TAM 10

// Prototipos
void Frecuencia( void *ptr);
void Moda(void *ptr);

//Variables compartidas
struct lims { int ini, fin;};
struct lims limites[2] = {{0,4},{5,9}};
int nums[TAM] = {1,4,7,2,9,5,6,3,1,2};
int frec[TAM] = {0,0,0,0,0,0,0,0,0,0};
int modas[TAM];
int k;

// Candados con los parámetros por defecto (abierto)
pthread_mutex_t mis_candados[TAM];

int main()
{
   int i;
   pthread_t hilo1, hilo2, hilo3;

   for(i=0;i<TAM;i++)
       pthread_mutex_init(&mis_candados[i], NULL);
```

```
    pthread_create(&hilo1,NULL,(void *) &Frecuencia, (void *)&limites[0]);
    pthread_create(&hilo2,NULL,(void *) &Frecuencia, (void *)&limites[1]);

    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

    pthread_create(&hilo3,NULL,(void *) &Moda,NULL);
    pthread_join(hilo3,NULL);

    printf("Hilo padre: las frecuencias son \n\n");
    for (i=0;i<TAM; i++)
        printf("[%d]:frec[%d]\n", i+1, frec[i]);

    i=0;
    printf("\nModa\t= ");
    while(modas[i]!=0){
        printf("%d ",i+1);
        i++;
    }

    printf("\n");
    // Se destruyen los candados
    for(i=0;i<TAM;i++)
        pthread_mutex_destroy(&mis_candados[i]);

    return 0;
}

void Frecuencia(void *ptr){
    int i,j,num,cont;

    struct lims *mis_limites = (struct lims *) ptr;

    for(num = 1; num<=10; num++){
                cont = 0;
                for(j= (mis_limites->ini); j<= (mis_limites->fin); j++)
```

```
                    if(nums[j] == num)
                        cont++;
                pthread_mutex_lock(&mis_candados[num-1]);
                frec[num-1] += cont;
                pthread_mutex_unlock(&mis_candados[num-1]);
    }
    pthread_exit(0);
}

void Moda(void *ptr){
    int i,j,max=0,moda;
    for (i=0;i<TAM-1;i++){
        moda=0;
        for(j=i+1;j<TAM;j++){
            if (nums[i] == nums[j]){
                moda++;
            }
        }
        if((moda>max) && (moda!=0)){
            k = 0;
            max = moda;
            modas[k] = frec[i];
            k++;
        }
        else if (moda==max){
            modas[k] = nums[i];
            k++;
        }
    }
}
```