



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

PROFESOR: RANGEL GONZALEZ JOSUE

APLICACIONES PARA COMUNICACIONES DE RED

3CV7

PRACTICA 2

ALUMNOS:

JOSUE ARTURO ALFARO BRACHO
VLADIMIR AZPEITIA HERNANDEZ
ALDO SUAREZ CRUZ

INTRODUCCIÓN

El presente trabajo sobre threads y semáforos forma parte de una de las bases para poder entender el funcionamiento de temas más complejos que iremos abordando durante este curso. Ya que en la práctica anterior ya abordamos a profundidad el concepto de thread en este caso nos enfocaremos más a la parte de los semáforos.

Los semáforos son una estructura diseñada para sincronizar dos o más threads o procesos, de forma que su ejecución se realice de forma ordenada y sin conflictos entre ellos.

El por qué no se pueden usar directamente otras estructuras más clásicas, como por ejemplo usar una variable común para decidir si se puede o no acceder a un recurso, se debe a que estamos en un sistema multitarea: hacer esto implicaría realizar una espera activa (un bucle, comprobando constantemente si la variable está o no a 0, y así saber si podemos seguir ejecutando o no). Por otro lado, puede ocurrir algo mucho peor: supongamos que un proceso comprueba la variable, y ve que el recurso está libre, por lo que procedería a cambiar dicha variable de valor y seguir. Pues bien, si justo después de la comprobación pero antes de que cambie el valor se conmuta de tarea (puede pasar, pues el sistema operativo puede hacerlo en cualquier momento), y el nuevo proceso comprueba la variable, como todavía no se ha actualizado, creerá que el recurso está libre, e intentará tomarlo, haciendo que ambos programas fallen. Lo peor del caso es que se tratará de un error aleatorio: unas veces fallará (cuando se produzca cambio de tarea en ese punto) y otras no.

Para evitarlo, se idearon los semáforos. Un semáforo básico es una estructura formada por una posición de memoria y dos instrucciones, una para reservarlo y otra para liberarlo. A esto se le puede añadir una cola de threads para recordar el orden en que se hicieron las peticiones, esta posición de memoria es mejor conocida como sección crítica.

CONCEPTOS

Sección crítica

Se denomina región crítica, (sección crítica y región crítica son denominaciones equivalentes) en programación concurrente de ciencias de la computación, a la porción de código de un programa de ordenador en la que se accede a un recurso compartido (estructura de datos o dispositivo) que no debe ser accedido por más de un proceso o hilo en ejecución.

Semáforos binarios

Un semáforo binario es un indicador de condición (S) que registra si un recurso está disponible o no. Un semáforo binario sólo puede tomar dos valores: 0 y 1. Si, para un semáforo binario, $S=1$ entonces el recurso está disponible y la tarea lo puede utilizar; si $S=0$ el recurso no está disponible y el proceso debe esperar.

Semáforos con múltiples variables

En este caso el semáforo se inicializa con el número total de recursos disponibles (n) y las operaciones de WAIT y SIGNAL se diseñan de modo que se impida el acceso al recurso protegido por el semáforo cuando el valor de éste es menor o igual que cero.

Cada vez que se solicita y obtiene un recurso, el semáforo se decrementa y se incrementa cuando se libera uno de ellos. Si la operación de espera se ejecuta cuando el semáforo tiene un valor menor que uno, el proceso debe quedar en espera de que la ejecución de una operación señal libere alguno de los recursos.

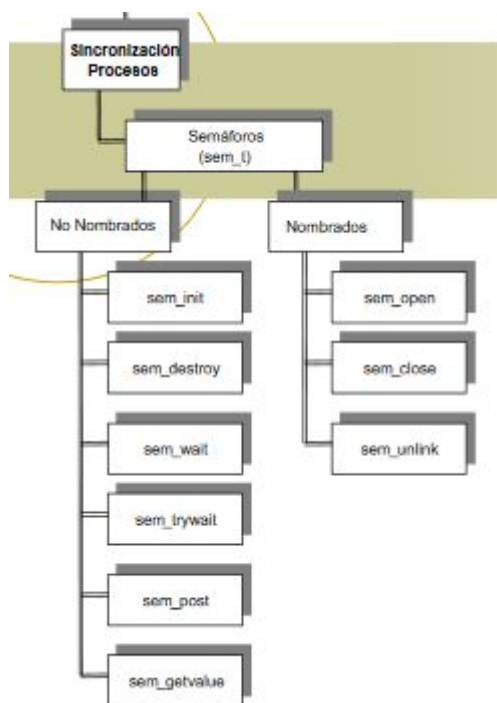
Al igual que en los semáforos binarios, la ejecución de las operaciones son indivisibles, esto es, una vez que se ha empezado la ejecución de uno de estos procedimientos se continuará hasta que la operación se haya completado.

DIFERENCIAS ENTRE SEMÁFORO BINARIO Y CON MÚLTIPLES VARIABLES

- La variable de un semáforo binario sólo tiene permitido tomar los valores 0 (ocupado) y 1 (libre) en cambio un semáforo con múltiples variables puede tomar cualquier valor entero, puede variar en un dominio no restringido.
- Semáforos binarios son más fáciles de poner en práctica en comparación con el semáforo con múltiples variables.
- Semáforo binario permite sólo un hilo para acceder al recurso a la vez, pero un semáforo con múltiples variables permite n hilos que acceden a la vez.

- Las 2 operaciones que se definen para los semáforos binarios son TAKE y RELEASE y las 2 operaciones que se definen para los semáforos con múltiples variables son WAIT y SIGNAL.
- El semáforo binario resulta adecuado cuando hay que proteger un recurso que pueden compartir varios procesos, pero cuando lo que hay que proteger es un conjunto de recursos similares, se puede usar un semáforo con múltiples variables para que lleve la cuenta del número de recursos disponibles.
- En los semáforos binarios muchos procesos pueden estar en suspensión en la lista, mientras que en los semáforos con múltiples variables los procesos avanzan sin demora.

Manejo de Semáforos en UNIX



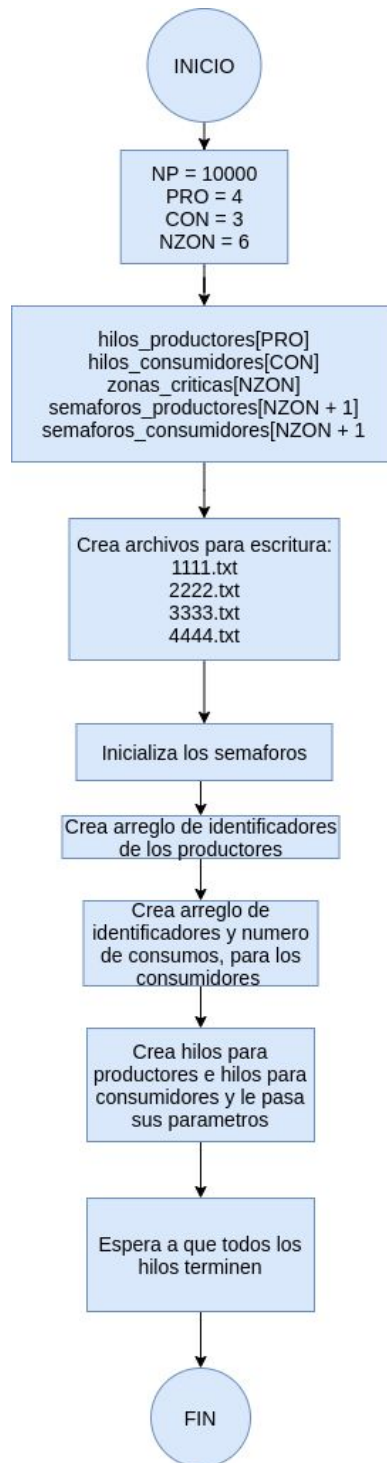
POSIX define dos tipos:

- Semáforos nombrados: se puede utilizar por el proceso que lo crea y por cualquier otro aunque no tenga relación con el creador
- Semáforos no nombrados: se puede utilizar por el proceso que lo crea (threads). El uso desde otros procesos precisa de una región de memoria compartida

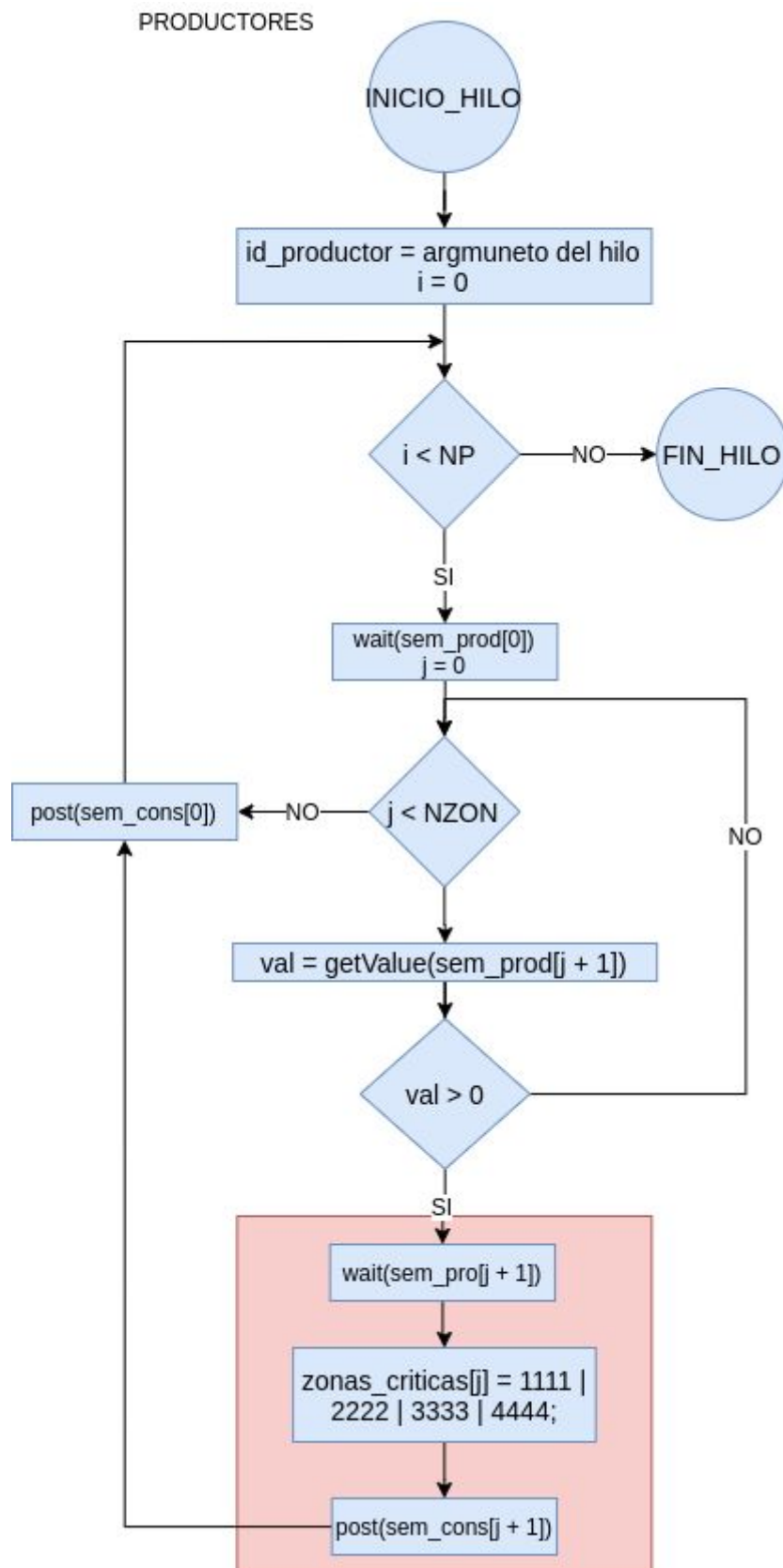
DESARROLLO

Esta práctica abordará el tema de threads y semáforos un un ejercicio en el cual se declaran 6 secciones críticas, 4 productores, 3 consumidores y 10000 producciones por cada productor, esta práctica se realizó en el lenguaje c.

Diagrama de Flujo

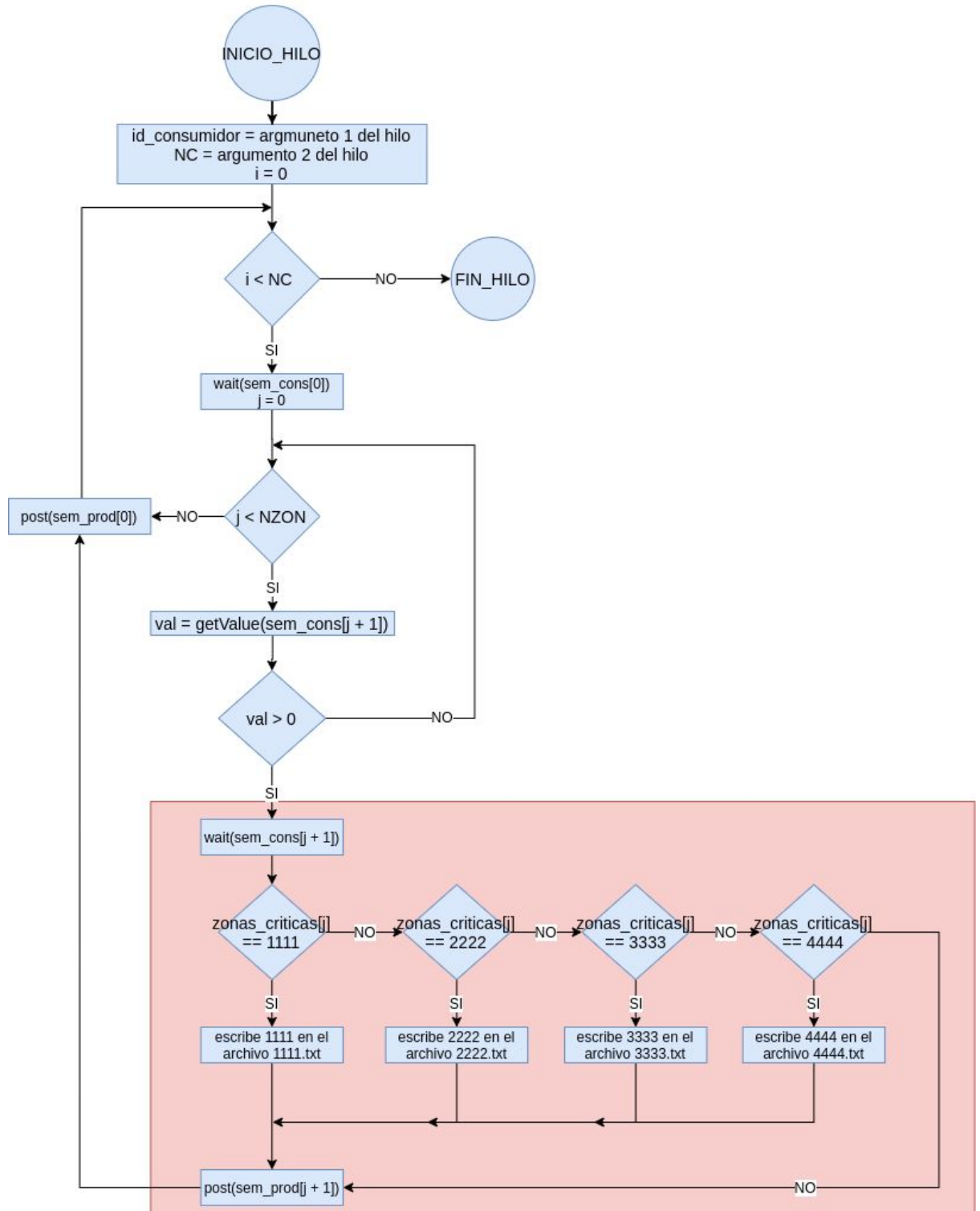


Hilo productor:



Hilo consumidor:

CONSUMIDORES



Código de Implementación

- ❑ Lo primero que se realiza es definir el número de repeticiones, secciones críticas, consumidores y productores junto con los includes de las librerías para utilizar pthreads y semáforos, a partir de eso también definimos los semáforos y nombramos las variables para los archivos que vamos a utilizar para imprimir la información

```
#include <pthread.h>    // Biblioteca para hilos POSIX
#include <semaphore.h>  // Biblioteca para semaforos POSIX
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NP 10000 // Numero de producciones (por cada productor)
#define PRO 4    // Numero de productores
#define CON 3    // Numero de consumidores
#define NZON 6   // Numero de zonas criticas

int buffer[NZON];           // Zonas Criticas
sem_t sem_prod[NZON + 1], sem_cons[NZON + 1]; // Semaforos

FILE *f1, *f2, *f3, *f4;    // Archivos
```

- ❑ Después pasamos a ver la función para los productores que será la siguiente como se puede ver en esta parte se realiza el manejo de los semáforos para las secciones críticas:

```
void *prod_thread(void *arg)
{
    int id_producer = *((int *) arg);
    for (int i = 0; i < NP; i++)
    {
        sem_wait(&sem_prod[0]);
        for (int j = 0; j < NZON; j++)
        {
            int val;
            sem_getvalue(&sem_prod[j + 1], &val);
            if (val > 0)
            {
                sem_wait(&sem_prod[j + 1]);
                buffer[j] = id_producer * 1111;
                //printf("P%d: Se produjo %d en la zona critica %d\n", id_producer, buffer[j], j + 1);
                sem_post(&sem_cons[j + 1]);
                break;
            }
        }
        sem_post(&sem_cons[0]);
    }
    printf("P%d: Termine mi trabajo\n", id_producer);
    pthread_exit(NULL);
}
```


- ❑ Para la parte de los consumidores utilizamos la siguiente función, como se puede ver en ambas funciones, están hechas de tal forma que sin importar que productor o consumidor entre se realice la acción solicitada:

```
void *cons_thread(void *arg)
{
    parameters params = *((parameters *) arg);
    int id_consumer = params.id;
    int max = params.bnum;
    for (int i = 0; i < max; i++)
    {
        sem_wait(&sem_cons[0]);
        for (int j = 0; j < NZON; j++)
        {
            int val;
            sem_getvalue(&sem_cons[j + 1], &val);
            if (val > 0)
            {
                sem_wait(&sem_cons[j + 1]);

                //printf("C%d: Se consumo %d en la zona critica %d\n", id_consumer, buffer[j], j + 1);
                switch (buffer[j]) // Escribir en archivos de texto
                {
                    case 1111:
                        fprintf(f1, "%s\n", "1111");
                        break;
                    case 2222:
                        fprintf(f2, "%s\n", "2222");
                        break;
                    case 3333:
                        fprintf(f3, "%s\n", "3333");
                        break;
                    case 4444:
                        fprintf(f4, "%s\n", "4444");
                        break;
                    default:
                        printf("C%d: Hubo un error al momento de consumir\n", id_consumer);
                        break;
                }
                buffer[j] = -1;
                sem_post(&sem_prod[j + 1]);
                break;
            }
        }
        sem_post(&sem_prod[0]);
    }
    printf("C%d: Termine mi trabajo\n", id_consumer);
    pthread_exit(NULL);
}
```

- ❑ En el main se tiene la inicialización de los archivos para escritura, de los semáforos y los hilos, también el llenado de las estructuras para almacenar id y cantidad de consumo para cada una dependiendo de los datos que se introduzcan para producciones, consumidores y productores.

```
int main()
{
    //Hilos
    pthread_t h_prod[PRO], h_cons[CON];

    f1 = fopen("1111.txt", "w"); //Archivo donde se guardan 1111
    f2 = fopen("2222.txt", "w"); //Archivo donde se guardan 2222
    f3 = fopen("3333.txt", "w"); //Archivo donde se guardan 3333
    f4 = fopen("4444.txt", "w"); //Archivo donde se guardan 4444

    // inicializacion de semaforos
    for (int i = 1; i < NZON + 1; i++)
    {
        sem_init(&sem_prod[i], 0, 1);
        sem_init(&sem_cons[i], 0, 0);
    }

    sem_init(&sem_prod[0], 0, NZON);
    sem_init(&sem_cons[0], 0, 0);

    //Arreglo de identificadores para productores y consumidores
    int ids_prod[PRO];
    for (int i = 0; i < PRO; i++)
    {
        ids_prod[i] = i + 1;
    }

    parameters params[CON];
    int div = floor(NP * PRO / CON);
    int mod = (NP * PRO) % CON;
    for (int i = 0; i < CON; i++)
    {
        params[i].id = i + 1;
        params[i].bnum = div;
        if(i == 0)
            params[i].bnum += mod;
    }
}
```

```

// creacion de hilos productores
for (int i = 0; i < PRO; i++)
{
    pthread_create(&h_prod[i], NULL, prod_thread, (void *)&ids_prod[i]);
}

// creacion de hilos consumidores
for (int i = 0; i < CON; i++)
{
    pthread_create(&h_cons[i], NULL, cons_thread, (void *)&params[i]);
}

// Esperar a que terminen los hilos
for (int i = 0; i < PRO; i++)
{
    pthread_join(h_prod[i], NULL);
}

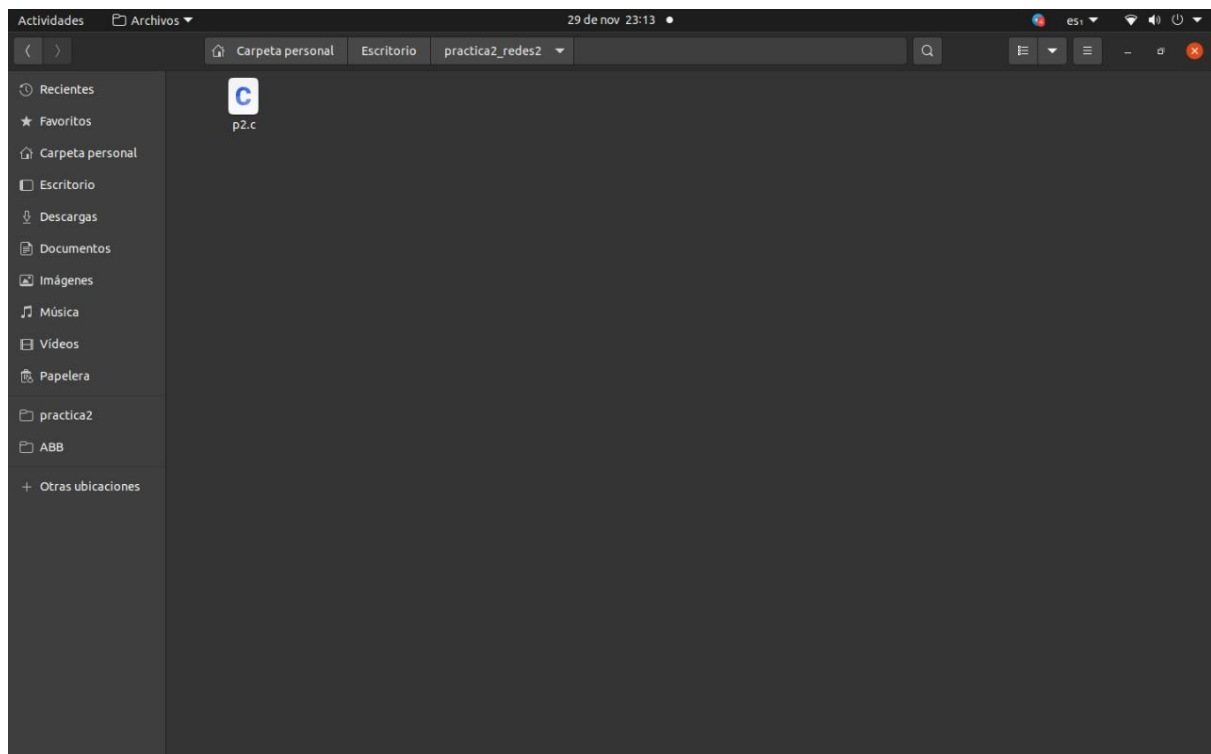
for (int i = 0; i < CON; i++)
{
    pthread_join(h_cons[i], NULL);
}

return 0;

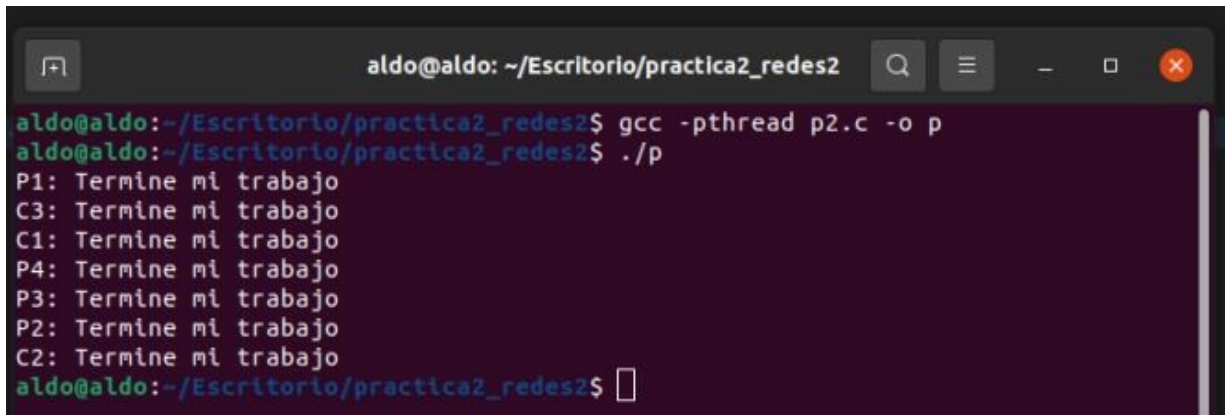
```

Ejecución

Nos ponemos en la carpeta donde está nuestro programa la cual antes de la ejecución se ve de la siguiente manera.

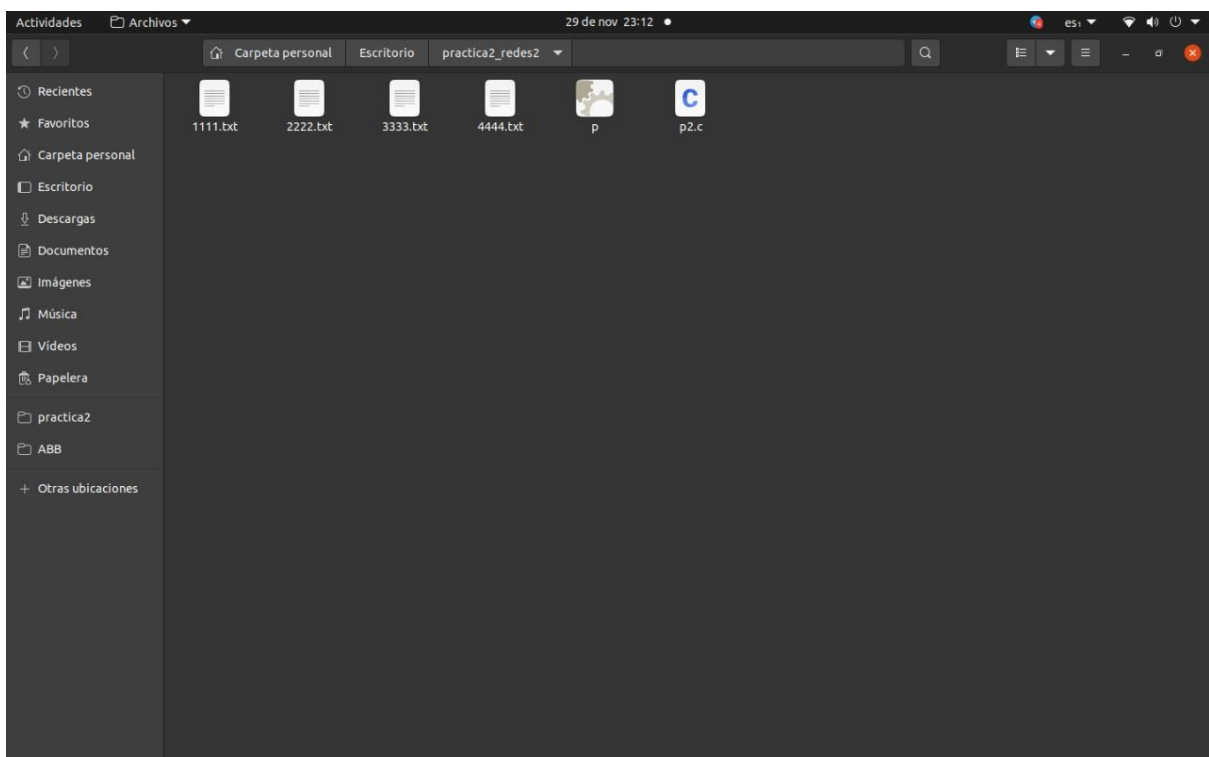


- ❑ Posteriormente entramos en consola y realizaremos la compilación y ejecución de el código, el cual imprime lo siguiente en consola:



```
aldo@aldo: ~/Escritorio/practica2_redes2
aldo@aldo:~/Escritorio/practica2_redes2$ gcc -pthread p2.c -o p
aldo@aldo:~/Escritorio/practica2_redes2$ ./p
P1: Termine mi trabajo
C3: Termine mi trabajo
C1: Termine mi trabajo
P4: Termine mi trabajo
P3: Termine mi trabajo
P2: Termine mi trabajo
C2: Termine mi trabajo
aldo@aldo:~/Escritorio/practica2_redes2$
```

- ❑ Una vez que termine la ejecución veremos como terminaron todos los productores y todos los consumidores y en la carpeta donde se encontraba el codigo se han creado los archivos de texto correspondientes a las producciones realizadas, el nombre de estos archivos se especifican en las funciones fopen .



- Podemos observar que el programa finalizó con éxito, ya que en cada archivo de texto están las 10000 producciones que se esperaban

```
File Edit Options Buffers Tools Text Help
1111.txt Bot (Text)
9978 1111
9979 1111
9980 1111
9981 1111
9982 1111
9983 1111
9984 1111
9985 1111
9986 1111
9987 1111
9988 1111
9989 1111
9990 1111
9991 1111
9992 1111
9993 1111
9994 1111
9995 1111
9996 1111
9997 1111
9998 1111
9999 1111
10000 1111

2222.txt Bot (Text)
9978 2222
9979 2222
9980 2222
9981 2222
9982 2222
9983 2222
9984 2222
9985 2222
9986 2222
9987 2222
9988 2222
9989 2222
9990 2222
9991 2222
9992 2222
9993 2222
9994 2222
9995 2222
9996 2222
9997 2222
9998 2222
9999 2222
10000 2222

3333.txt Bot (Text)
9978 3333
9979 3333
9980 3333
9981 3333
9982 3333
9983 3333
9984 3333
9985 3333
9986 3333
9987 3333
9988 3333
9989 3333
9990 3333
9991 3333
9992 3333
9993 3333
9994 3333
9995 3333
9996 3333
9997 3333
9998 3333
9999 3333
10000 3333

4444.txt Bot (Text)
9978 4444
9979 4444
9980 4444
9981 4444
9982 4444
9983 4444
9984 4444
9985 4444
9986 4444
9987 4444
9988 4444
9989 4444
9990 4444
9991 4444
9992 4444
9993 4444
9994 4444
9995 4444
9996 4444
9997 4444
9998 4444
9999 4444
10000 4444
```

CONCLUSIONES

ALDO SUAREZ CRUZ: Los hilos a pesar de ser muy útiles para la ejecución de procesos en ocasiones necesitan ser coordinados para la realización de tareas, en la práctica pasada se usaron hilos con funciones básicas, en esta práctica, los hilos creados fueron coordinados por medio de semáforos con los cuales se controlaban los accesos a zonas críticas para consumo o almacenar las producciones realizadas de esta forma no se sobreescriben datos y se tiene un mejor control sobre zonas críticas, la utilización de las zonas críticas se realiza de manera dinámica, de esta forma se evitan las asignaciones estáticas. La forma más práctica de solucionar el problema de diferente número de productores y consumidores es asignando un número de consumos dependiendo de la cantidad de productores y consumidores que se tenga.

JOSUÉ ARTURO ALFARO BRACHO: Con esta práctica la principal dificultad que se tuvo era la parte de sincronizar los semáforos para que dependiendo de lo que se realice se habilite o se deshabiliten las zonas críticas y que no exista un punto muerto, es parte a mi se me dificultó mucho en lo personal ya que no me lo imaginaba al inicio hasta irlo realizando,.

VLADIMIR AZPEITIA HERNÁNDEZ: Para esta práctica hay 4 productores, 3 consumidores y 6 secciones críticas. El problema que tuvimos fue al momento de asignar el número de consumos que iba a realizar cada consumidor. Ya que el número de consumidores es diferente a la de productores. Para solucionarlo pasamos un segundo parámetro a los hilos consumidores, y este fue el número de consumos que iba a realizar cada hilo, esta carga la distribuimos uniformemente y asignándole siempre la mayor carga al primer consumidor. La práctica cumple con los requisitos especificados, es decir que los productores y consumidores pueden acceder a cualquier zona crítica disponible, sin esperar por una asignada de manera estática.