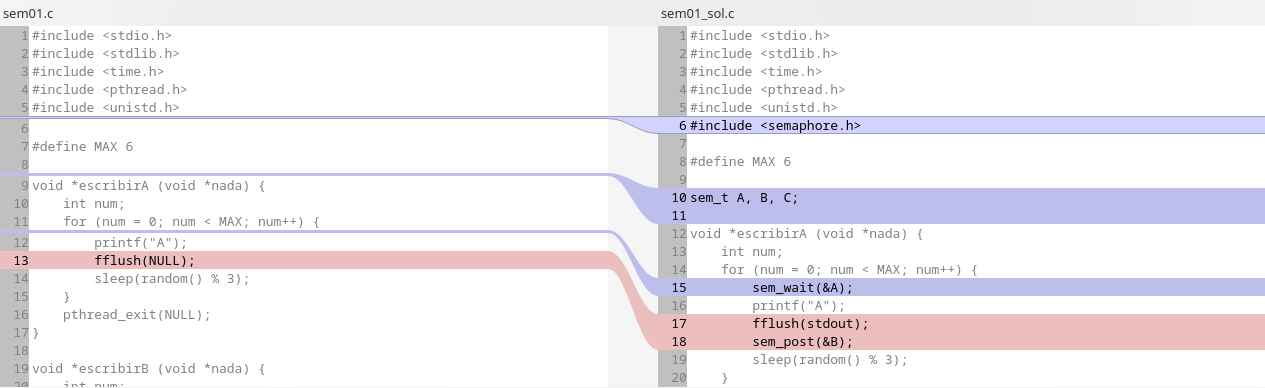
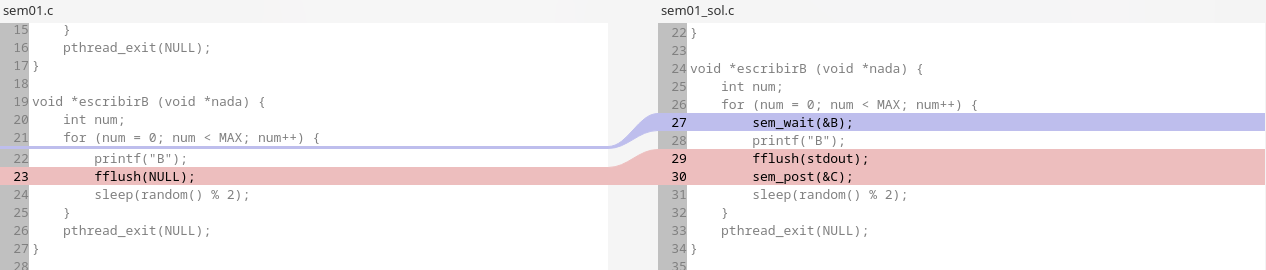
# 1. Ejercicio 1 – resuelto

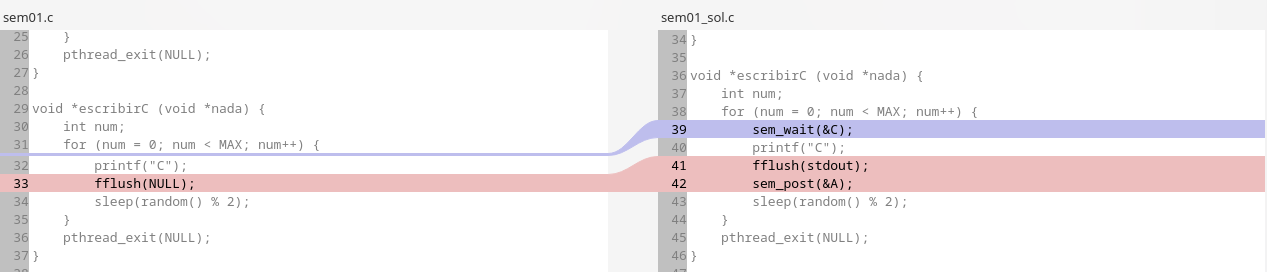
Se crean tres hilos de manera que uno ejecuta escribirA, otro escribirB y el tercero escribirC. Introduce los semáforos oportunos para que la salida sea ABCABCABCABCABCABC.

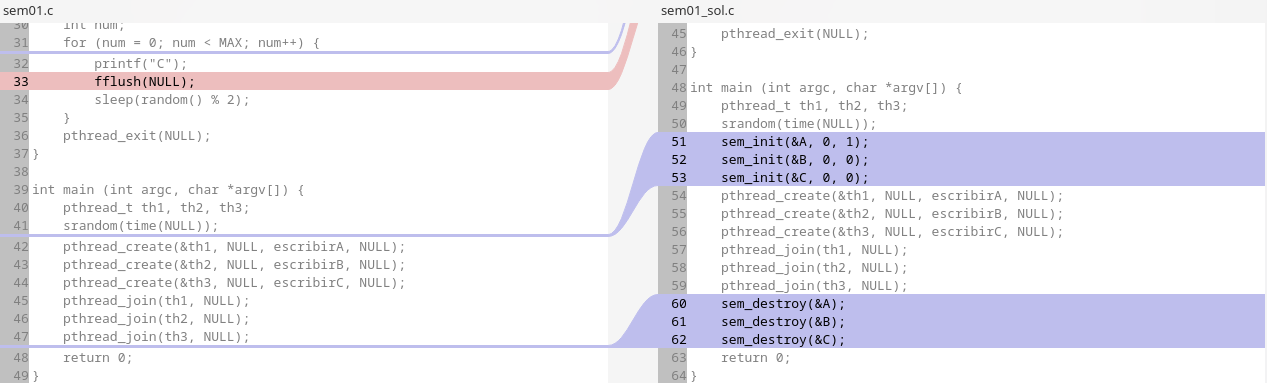
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <pthread.h>  #include <unistd.h>  #define MAX 6  void \*escribirA (void \*nada) {  int num;  for (num = 0; num < MAX; num++) {  printf("A");  fflush(NULL);  sleep(random() % 3);  }  pthread\_exit(NULL);  }  void \*escribirB (void \*nada) {  int num;  for (num = 0; num < MAX; num++) {  printf("B");  fflush(NULL);  sleep(random() % 2);  }  pthread\_exit(NULL);  }  void \*escribirC (void \*nada) {  int num;  for (num = 0; num < MAX; num++) {  printf("C");  fflush(NULL);  sleep(random() % 2);  }  pthread\_exit(NULL);  }  int main (int argc, char \*argv[]) {  pthread\_t th1, th2, th3;  srandom(time(NULL));  pthread\_create(&th1, NULL, escribirA, NULL);  pthread\_create(&th2, NULL, escribirB, NULL);  pthread\_create(&th3, NULL, escribirC, NULL);  pthread\_join(th1, NULL);  pthread\_join(th2, NULL);  pthread\_join(th3, NULL);  return 0;  } |

**Solución**. Modificar el anterior código de tal forma que se incluyan semáforos, como se puede apreciar en las siguientes capturas de pantalla (El código a la izquierda es el propuesto en el ejercicio y el de la derecha incluye las instrucciones de semáforos).







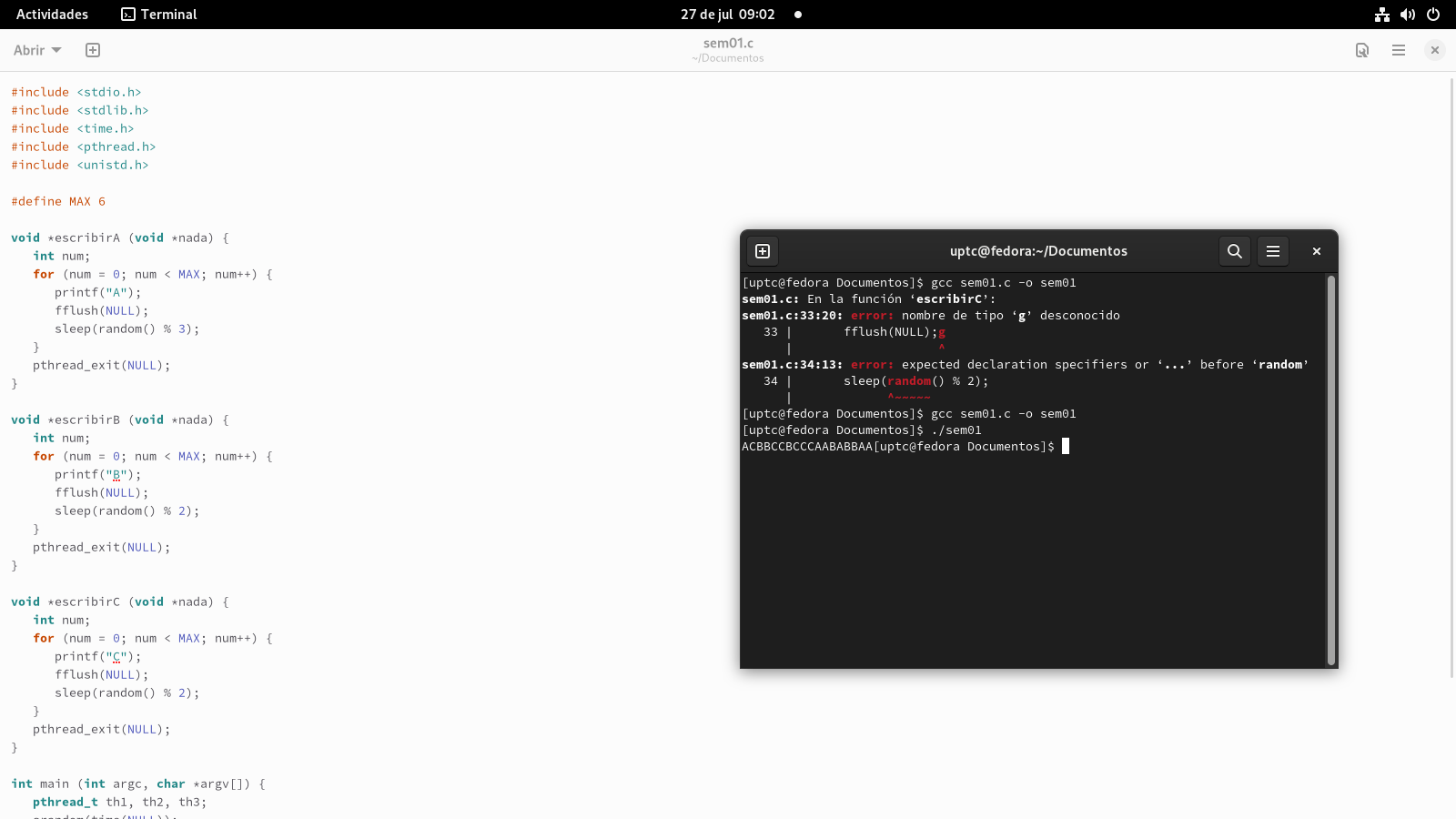


Desarrollo, para este punto, se siguió el paso a paso de la guia donde realizamos las correcciones que se nos sugerían en la solución dada, donde lo importante es resaltar los siguientes cambios:

* Se incluye la librería de semáforos con #define <semaphore.h>
* Se crearon 3 variables que nos representaran los semáforos para cada letra con sem\_t A, B, C
* Se agregó el método wait para los semáforos luego del for en cada uno de los métodos, es decir, en escribirA, escribirC y escribirC.
* En el método fflush de cada uno de los métodos, se le cambio el parámetro de NULL a stout
* Se agregó el método sem\_post, en cada uno de los métodos luego del fflush.
* En el main se inicia tres semáforos con sem\_init, pasandole los siguientes atributos:
  + &A, 0, 1
  + &B, 0, 0
  + &C, 0, 0
* Finalmente se crea los métodos de destroy para cada uno de los semáforos luego de los métodos de los hilos en el main

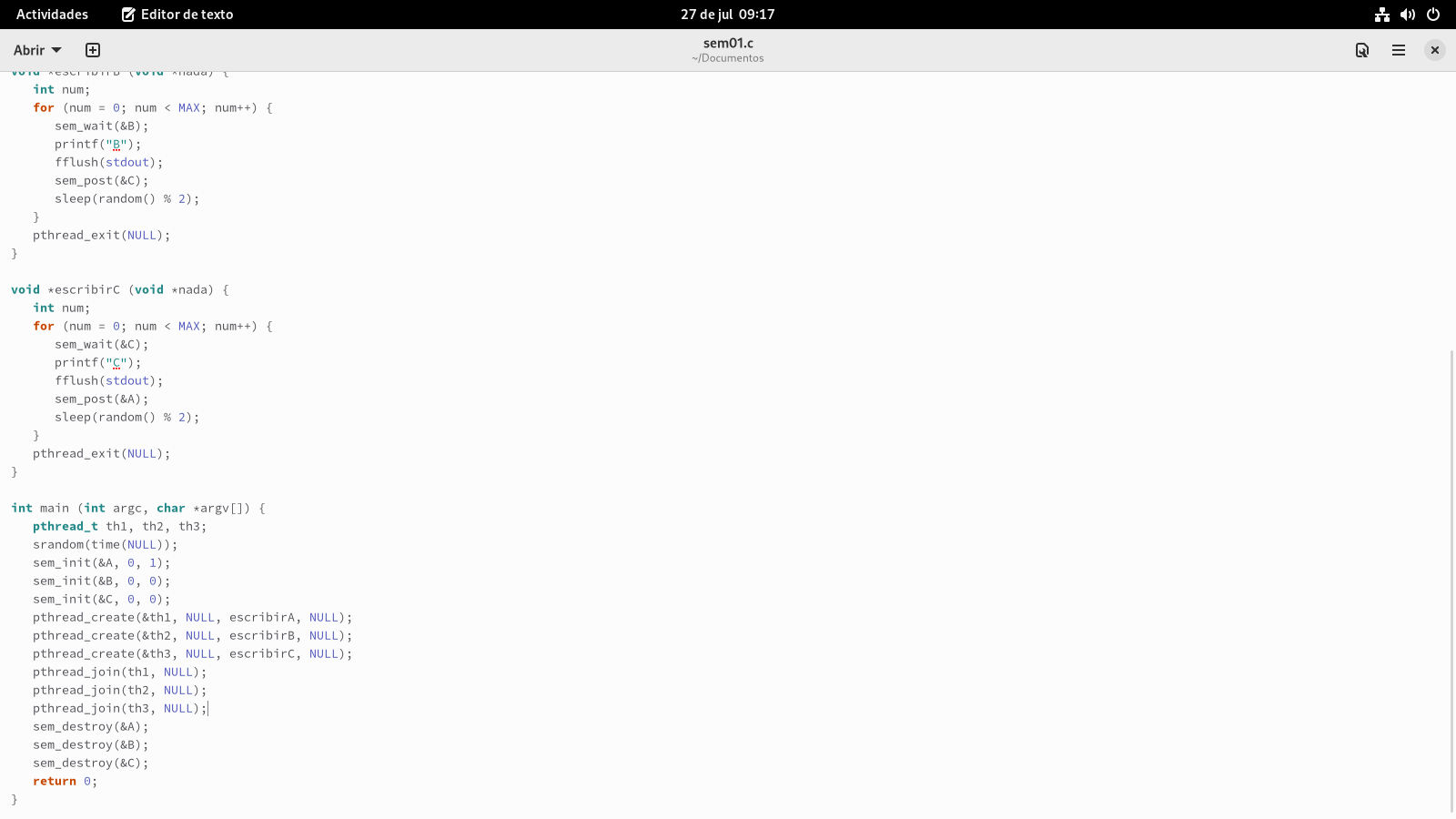
Este proceso anterior, lo realizamos de la manera en que se ve en las siguientes imagenes:

**Figura Código de solución para el ejercicio número 1, pt 1**



*Nota: autoría propia*

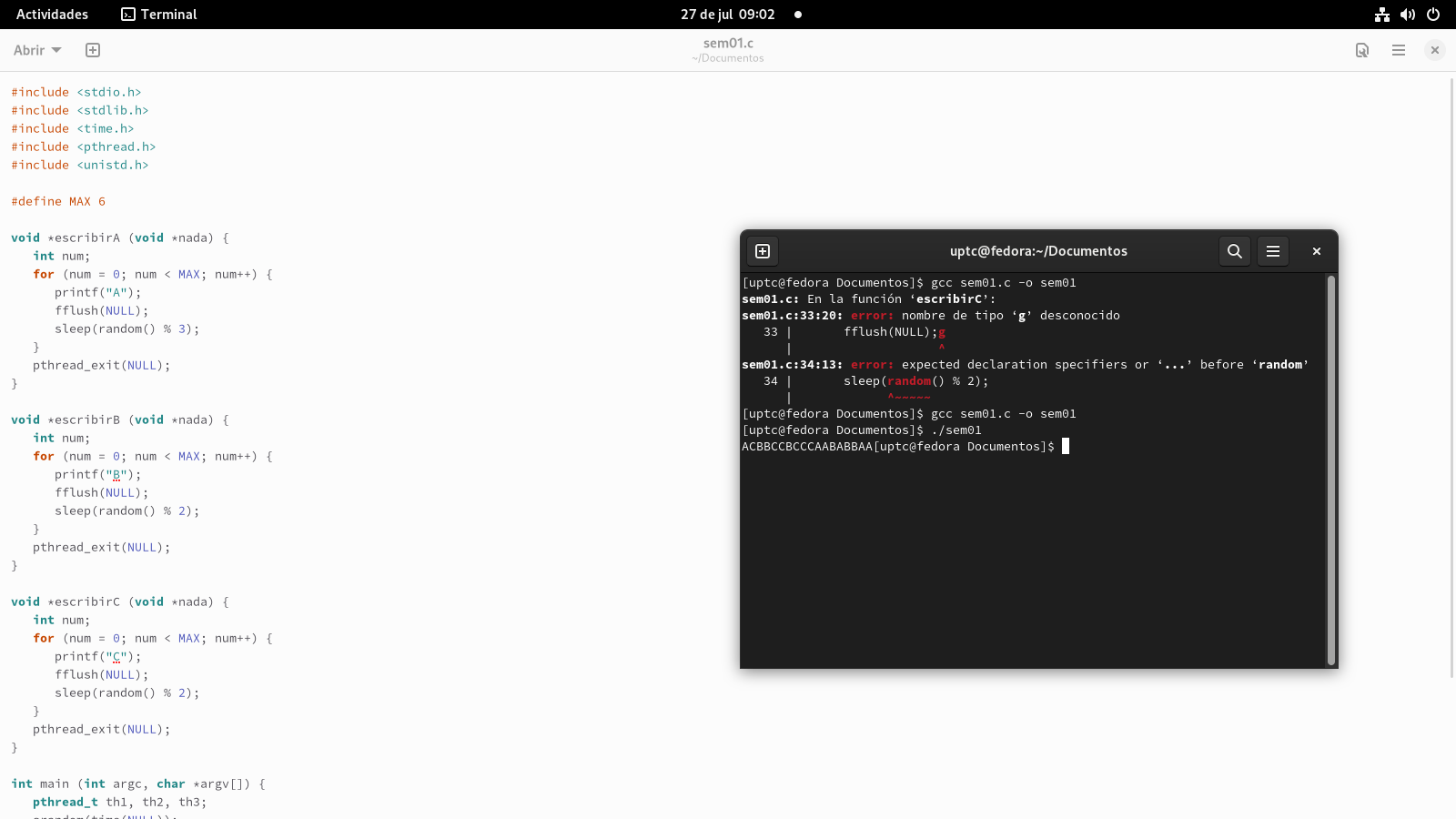
**Figura Código de solución para el ejercicio número 1, pt 2**



*Nota: autoría propia*

Luego de compilar, y corregir un error pequeño relacionado con una letra de más después de un punto y coma, se pasó a ejecutar, obteniendo el resultado esperado, como se muestra en la siguiente figura:

**Figura. Compilacion y ejecucion del ejercicio 1**



*Nota: autoría propia*

# 2. Ejercicio 2

Observa el siguiente fragmento de código donde los semáforos sem1 y sem2 están inicializados a cero, un hilo ejecuta la función incrementa y otro la función decrementa. Completa el programa, ejecútalo y describe los valores que, durante la ejecución, puede adoptar la variable num, así como las posibles situaciones de inter-bloqueo que pudieran darse.

|  |
| --- |
| int num = 10;  void \*incrementa(void \*nada) {  int i;  for (i = 0; i < 3; i++) {  sem\_wait(&sem1);  num++;  printf("Inc. Número = %d\n", num);  sem\_post(&sem1);  }  sem\_post(&sem2);  sleep(random() % 3);  sem\_wait(&sem2);  pthread\_exit(NULL);  }  void \*decrementa(void \*nada) {  int i;  for (i = 0; i < 3; i++) {  sem\_post(&sem1);  sleep(random() % 3);  sem\_wait(&sem2);  num--;  printf("Dec. Número = %d\n", num);  sem\_post(&sem2);  sem\_wait(&sem1);  }  sem\_wait(&sem1);  pthread\_exit(NULL);  } |

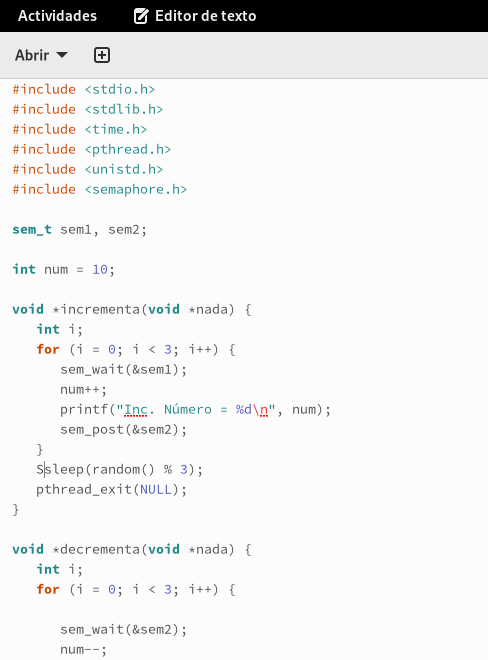
Con respecto a este punto, en primer lugar, incluimos la librerías necesarias para la ejecucion del codigo de este punto, donde incluimos stdio.h, stdlib.h, time.h, pthread.h, unistd.h y semaphore.h

Luego de ello, creamos dos variables llamadas sem1 y sem2 con sem\_t

Nos dimos cuenta que el los métodos de incrementar y decrementar había más de un sem\_wait y de un sem\_post, por lo cual pasamos a corregir ese aspecto, y dejamos únicamente los necesarios.

Finalmente, creamos el método main que hacía falta, donde iniciamos dos semáforos, dos hilos y finalmente destruiamos los semáforos, esto con sus respectivos métodos sem\_init, pthread\_create, pthread\_join y sem\_destroy, este procedimiento lo podemos ver en las siguiente figuras:

**Figura. Código de solución al punto 2 parte 1**



*Nota: autoría propia*

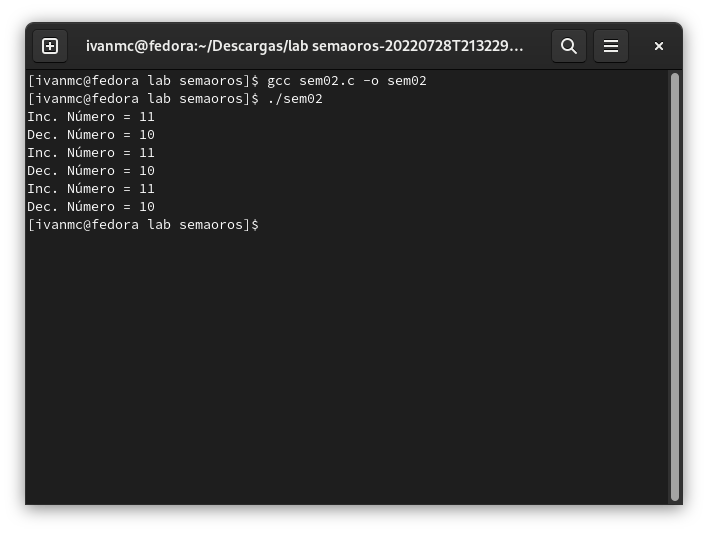
**Figura. Código de solución al punto 2 parte 2**

****

*Nota: autoría propia*

Finalmente, compilamos y ejecutamos el código de este punto, obteniendo el resultado esperado, como se muestra en la siguiente figura:

**Figura. Compilacion y ejecucion del código del punto 2**



*Nota: autoría propia*

# 3. Ejercicio 3

Se crean dos hilos de manera que uno ejecuta escribirA y el otro escribirB. Introduce los semáforos oportunos para que la salida sea BABABABABA. Completa el programa, ejecútalo. No olvides indicar los valores iniciales de los semáforos que utilices.

|  |
| --- |
| void \*escribirA (void \*p) {  int i;  for (i = 0; i < 5; i++) {  printf("A");  fflush(NULL);  sleep(random() % 3);  }  pthread\_exit(NULL);  }  void \*escribirB (void \*p) {  int i;  for (i = 0; i < 5; i++) {  printf("B");  fflush(NULL);  sleep(random() % 2);  }  pthread\_exit(NULL);  } |

Con respecto a este punto, en primer lugar, incluimos la librerías necesarias para la ejecucion del codigo de este punto, donde incluimos stdio.h, stdlib.h, time.h, pthread.h, unistd.h y semaphore.h

Luego de ello, creamos dos variables llamadas A y B con sem\_t

Nos dimos cuenta que el los métodos de escribirA y escribirB faltaba incluir los métodos de sem\_wait y sem\_post, de manera similar al punto 1, de igual forma en el método fflush, se cambio el parámetro de NULL a stdout

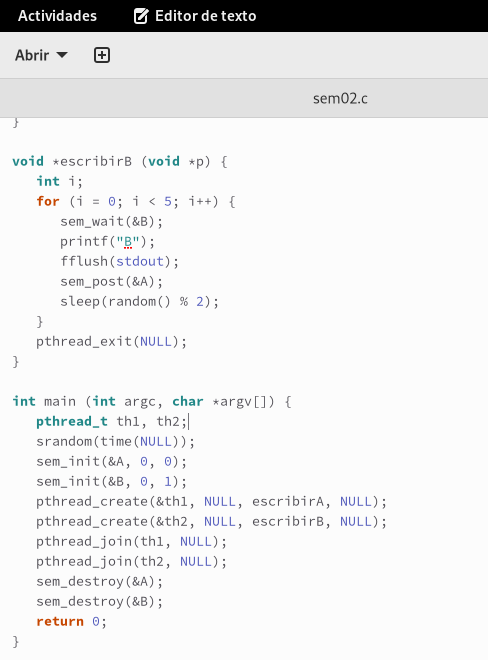
Finalmente, creamos el método main que hacía falta, donde iniciamos dos semáforos, dos hilos y finalmente destruiamos los semáforos, esto con sus respectivos métodos sem\_init, pthread\_create, pthread\_join y sem\_destroy, este procedimiento lo podemos ver en las siguiente figuras:

**Figura. Código de solución al punto 3 parte 1**



*Nota: autoría propia*

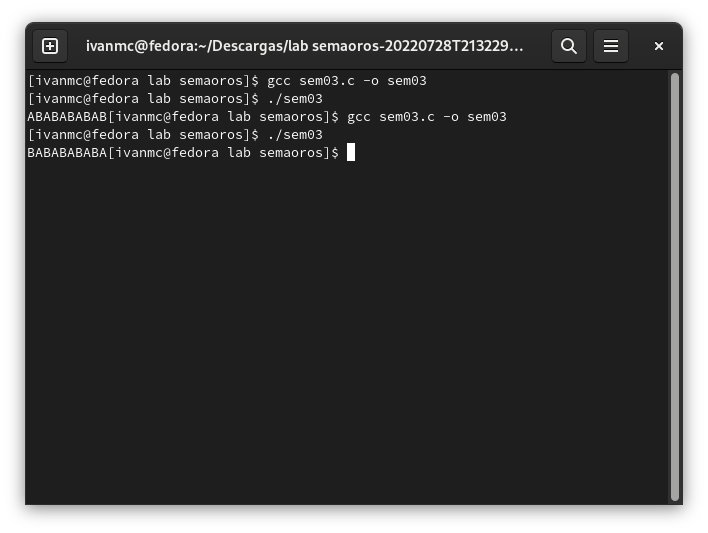
**Figura. Código de solución al punto 3 parte 2**



*Nota: autoría propia*

Finalmente, compilamos y ejecutamos el código de este punto, obteniendo el resultado esperado, como se muestra en la siguiente figura:

**Figura. Compilacion y ejecucion del código del punto 3**



*Nota: autoría propia*

# Conclusiones

* El orden de los hilos o el orden de los métodos sem\_wait y sem\_post afectan el resultado final, debido a que se puede bloquear o desbloquear un semáforo en el momento inadecuado, afectado la operación.
* Notamos que al tener más de un método sem\_wait y sem\_post en un método, este llegaba a afectar el resultado final esperado.
* Notamos cómo es que los semáforos pueden llegar a facilitar la planificación de los procesos en un sistema operativo, ya que se entiende que los semáforos permiten el tomar decisiones de manera sincrónica para ejecutar dos o más procesos

# Referencias

[1]. A. S. Tanenbaum, *Modern Operating Systems: Pearson New International Edition*. Pearson Education, Limited, 2013.