**Nombre:** Issac Alexander Maza Punine
**Materia:** Programación y Sistemas.
**Paralelo Nro:** 1
**Tarea # 4**

# Ejercicio 1

En total se tienen 8 posibles combinaciones (**A activado, D desactivado**):

1) **D-Join_Hilo1, D-Join_Hilo2, D-return(0)**
   En esta prueba con las 3 líneas comentadas, tuve dos situaciones.
   - La primera fue que el programa se congele o nunca termine. Mi explicación es que esto puede deberse al bloqueo mutuo(deadlock) en el código, principalmente causado por la falta de sincronización adecuada entre los hilos, lo que puede indicar que los hilos estén esperando entre sí en un orden permite que el programa avance. El hilo principal espera al que hilo 2, pero el hilo 2 nunca termina porque también esta esperando a que algo en el proceso del hilo1 suceda. O que ningún hilo pueda continuar su ejecución porque esta en un ciclo de espera mutua.

   - La segunda situación, ya que un resultado:

| | | | |
|---|---|---|---|
| 2: 4<br>0: 4<br>1: 5 | 2:4<br>0: 4<br>1: 5 | 2: 4<br>0: 4<br>1: 5 | 2: 4<br>0: 4<br>1: 5 |
| 2: 4<br>0: 4<br>1: 5 | 2: 4<br>0: 4<br>1: 5 | 2: 4<br>0: 4<br>1: 5 | 2: 4<br>0: 4<br>1: 5 |

   En este caso el hilo principal (main) no espera a que los hilos creados (h1 y h2) termine antes de continuar. No solo eso los valores de x impresos en el main pueden ser inconsistentes. No solo eso la ejecución de los hilos puede variar como se muestra, el sistema tiene una preferencia de comenzar y terminar el hilo 2, luego imprimir 0 : 4, finalmente imprimir el hilo 1.

   En estos dos casos el programa no usa explícitamente return(0) para indicar el final de main, en C, el programa aun puede finalizar correctamente.
2) **A-Join_Hilo1, D-Join_Hilo2, D-return(0)**
   En esta prueba se activo o descomento : pthread_join(myThread1, NULL); el programa principal(main) espera a que el hilo 1(h1) termine antes de continuar. Pero el hilo 2 (h2) no esta sincronizado, lo que puede indicar que a ala hora de ejecutarse podría o no terminar antes de que main imprima el valor de x.
   Algo seguro es que el hilo 1 siempre terminara antes de que main imprima el valor 0: x, pero el hilo 2 no, puede terminar antes que termine el h1, después del h1 , después de main o quizás nunca se ejecuta. En esta prueba se obtuvo lo siguientes resultados:

| | | | | |
|---|---|---|---|---|
| 2: 4 | 2: 4 | 2: 4 | 1: 8 | 2: 5 |

| | | | | |
|---|---|---|---|---|
| 1: 5<br>0: 5 | 1: 5<br>0: 5 | 1: 5<br>0: 5 | 2: 5<br>0: 5 | 1: 8<br>0: 5 |
| 1: 8<br>2: 5<br>0: 5 | 2: 4<br>1: 5<br>0: 5 | 1: 5<br>2: 4<br>0: 5 | 2: 4<br>1: 5<br>0: 5 | 2: 4<br>1: 5<br>0: 5 |

Cuando solo habilitamos el pthread_join del hilo1, el hilo principal espera a que h1 termine antes de continuar. Sin embargo, el hilo 2(h2) esta de alguna manera no esta sincronizado, por lo que la ejecución depende del sistema en cuando darle prioridad, así como también si se ejecuta de manera parcial, para luego terminar de ejecutarlo en otro momento. Esto genera resultados diferentes.
Explicación del resultado más frecuente:
2:4
1:5
0:5
El hilo 2 se ejecuta primero modificando x restando i que inicialmente es 2:
i++; //i=3
x = x -i; //X = 7 -3 = 4
Imprime 2 : 4
El hilo 1 (h1) se ejecute después, modifica x sumando i (que ahora es 3):
i = 1; // i se reinicia a 1;
x = x + i; // x = 4 +1 = 5
i++; //i = 2
Imprime 1: 5
El hilo principal (main) imprime el valor de x después de esperar a h1 = 0:5

También hay ejecución donde los hilos terminar antes que main termine, pero la ejecución de los hilos nos es de un solo paso sino que se ejecutan cada uno en dos pasos:

| |
|---|
| 1 : 8<br>2 : 5<br>0 : 5 |

3) **D-Join_Hilo1, A-Join_Hilo2, D-return(0)**
**En este caso solo habilitamos el pthread_join del hilo2, es bastante similar al caso 2, pero con la diferencia es que siempre se ejecutará el hilo 2, el hilo principal (main) esparará a que el hilo2 (h2) termine antes de continuar, mientras que el hilo 1(h1) seguirá ejecutándose de forma indepiente, ya sea antes que el hilo2, después del h2, después del hilo main, así como puede que nuca se ejecute.**

| **Resultado más frecuente** | **Resaultado menos frecuente** |
|---|---|
| **2 : 4**<br>**1: 5**<br>**0: 5** | **1: 8**<br>**2: 5**<br>**0: 5** |

**Esto ocurre porque varios factores:**
Sincronización parcial: al activar solo el join del hilo 2, el hilo principal espera a que h2 termine antes de continuar, pero el hilo1 no esta sincronizado con el hilo principal, por lo que puede ejecutarse antes, durante o después de h2.

El sistema da propiedades al hilo que cree que merece prioridad, el orden de ejecución de los hilos depende el sistema operativo. Esto explica por qué a veces h1 se ejecuta primero(1:8) y otra veces lo hace h2(2:4).
Acceso concurrente a variables compartidas, tanto h1 y h2 acceden a y cambian las mismas variables (x e i) sin exclusión mutua(mutex), generando resultados dependientes del momento en que cada hilo accede a dichas variables.

**En el caso a: 2:4, 1:5, 0:5**
**El hilo** 2 se ejecuta primero, modifica x restando i(i=2):
i++; -> i= 3
x = x -i; -> x = 7-3 = 4
Imprime 2: 4
**El hilo 1** se ejecuta después de que el hilo principal ya esperó a h2:
i = 1;-> i se reinicia a 1
x= x +i; -> x = 4+1=5
**El hilo principal imprime el valor actual de x: 0:5**
**En el caso b:** 1:8, 2: 5, 0:5
**El hilo1** se ejecuta primero:
i = 1; //I se reinica a 1
x = x +i; // x = 7 +1 = 8
i++; // i= 2
Imprime 1 :8
**El hilo 2 se ejecuta después:**
i++; // i = 3
x = x -i; // x = 8 -3 =5
Imprime 2 : 5
**El hilo principal** imprime el valor actual de x: 0:5

4) **D-Join_Hilo1, D-Join_Hilo2, A-return(0)**
En este caso al no usar pthread_join el hilo principal no espera a que los hilos h1 y h2 terminen antes de continuar con su ejecución, puedene también hacer que los hilos 1 y 2 se interrumpan debido a que ya termino el programa o los mensajes de los hilos podrían no apareces o parecer de manera no sincronizada eso depende si los hilos logran ejecutarse antes de que el programa termine.
**En esta prueba se obtuvieron los siguientes resultados:**

| | | | |
|---|---|---|---|
| **Caso Común**<br>**2: 4**<br>**1: 4**<br>**0: 5** | **Caso menor frecuente**<br>**1: 8**<br>**0:7**<br>**2:5** | **Caso raro**<br>**0:7**<br>**2:4**<br>**1: 5** | **Caso comun**<br>**2:4**<br>**0:4**<br>**1:5** |
| **Caso raro**<br>**0:7**<br>**1:5**<br>**2:4** | **Caso raro**<br>**0:7**<br>**1:8**<br>**2:5** | **2:4**<br>**0:4**<br>**1:5** | **2:4**<br>**0:4**<br>**1:5** |

Esto debe falta de sincronización, sin pthread_join el hilo principal no espera a que h1 y h2 terminen antes de ejecutar return(0). Provocando una serie de ejecuciones de los hilos sea impredecible.

Terminación del programa, aunque activase return(0); el programa no termia inmediatamente porque los hilos 1 y h2 ya están ejecutándose en segundo plano. Sin embargo, el hilo principal 0: sin esperar los resultados de los hilos.
También el sistema operativo le da prioridad a lo que el considera importante**.**

5) **A-Join_Hilo1, A-Join_Hilo2, D-return(0)**
Este es un caso deseable ya que al habilitar las líneas pthread_join del hilo 1 e hilo 2 terminen antes de que continue el hilo principal, haciendo que el comportamiento será mucho más consistente.
Resultados:

| | | | |
|---|---|---|---|
| 1:8<br>2:5<br>0:5 | 2:5<br>1:8<br>0:5 | 2:4<br>1:5<br>0:5 | 2:4<br>1:5<br>0:5 |
| 1:8<br>2:5<br>0:5 | 2:4<br>1:5<br>0:5 | 1:8<br>2:5<br>0:5 | 1;5<br>2:4<br>0:5 |

Al usar ambos pthread_join, el valor final de x impreso por el hilo principal siempre sea el resultado de todas las modificaciones realizadas por los hilos. Esto eliminara resultados impredecibles como 0:7 o 0: 4 vistos en pruebas anteriores.
A la hora de mostrar los mensajes, puede variar un poco el orden entre 1: y 2: , pero el valor final de x siempre será consistente(0:5), ya que reflejara las modificaciones de x de ambos hilos. Aunque eso si sin mecanismos de exclusión mutua(mutex), las modificaciones a x y i pueden ocurrir de manera concurrente, lo que podría generar resultados poco comunes pero posibles.

6) **A-Join_Hilo1, D-Join_Hilo2, A-return(0)**
En este situación donde solo esta comentada el pthread_join del hilo 2, no es muy diferente a los 2 y 3. Y estos fueron los resultados:

| | | | | | |
|---|---|---|---|---|---|
| **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **1:8**<br>**2:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **1:8**<br>**2:5**<br>**0:5** |
| **2:4**<br>**1:5**<br>**0:5** | **1:5**<br>**2:4**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** |

Como se puede ver al activar pthread_join del hilo 1 y la línea return(0); su comportamiento será similar al caso en que se activase únicamente el pthread_join del hilo 1, pero con algunas diferencias sutiles debido a la finalización explicita del programa con return(0);
El hilo principal(main) espera al hilo 1:Pthread_join(myThread1, NULL); garantiza que el hilo principal no continue hasta que el hilo 1 haya terminado.
El hilo 2 no esta sincronizado, esto se ve por el hilo principal que no espera a que el hilo 2, por lo que su ejecución depende del sistema operativo le de prioridad, esto genera diferentes resultados basados en cuando se ejecuta h2.
El programa finaliza después de que h1 termina, el hilo principal imprime el valor de x y termina con return(0).

7) **D_Join_Hilo1, A-Join_Hilo2, A-return(0)**

Esto tampoco difiere de los primeros casos, y al caso anterior, solo difiere que el programa si o si esperara a que el pthread_join del hilo 2 siempre se ejecute antes de que el programa continue, el hilo principal no espera al hilo 1, por lo que puede o no terminar antes de que el programa finalice y si el hilo principal finaliza el cual se envié el mensaje x:  y no termina h1, el hilo 1 será interrumpido.

| | | | | | |
|---|---|---|---|---|---|
| 2:4<br>1:5<br>0:5 | 2:4<br>1:5<br>0:5 | 2:4<br>1:5<br>0:5 | 1:8<br>2:5<br>0:5 | 1:8<br>2:5<br>0:5 | 2:4<br>1:5<br>0:5 |
| 2:4<br>1:5<br>0:5 | 1:5<br>2:4<br>0:5 | 2:4<br>1:5<br>0:5 | 1:8<br>2:5<br>0:5 | 2:4<br>1:5<br>0:5 | 2:4<br>1:5<br>0:5 |

**Explicación del caso común: 2: 4, 1: 5, O: 5**

1. El hilo 2 (h2) comienza su ejecución:
    - Incrementa i de 2 a 3.
    - Modifica x: x = 7 - 3 = 4.
    - Imprime: 2: 4.
2. El hilo 1 (h1) se ejecuta después de que el hilo principal imprime O::
    - Reinicia i a 1.
    - Modifica x: x = 4 + 1 = 5.
    - Imprime: 1: 5.
3. El hilo principal imprime el valor de x modificado por ambos hilos:

    - 0: 5

**Explicación del caso menos frecuente: 1: 8, 2: 5, O: 5**

1. El hilo 1 (h1) se ejecuta primero:
    - Reinicia i a 1.
    - Modifica x: x = 7 + 1 = 8.
    - Imprime: 1: 8.
2. El hilo 2 (h2) se ejecuta después:
    - Incrementa i de 1 a 3.
    - Modifica x: x = 8 - 3 = 5.
    - Imprime: 2: 5.
3. El hilo principal imprime el valor final de x:

    - 0: 5

**Explicación del comportamiento:**

1. El comportamiento del programa es que siempre espera que el hilo 2 termine antes de continuar. Por lo tanto:
    - 2:..... siempre se imprime antes que 0:.....
    - Sin embargo, el hilo 1 no esta sincronizado con el hilo principal, lo que genera variaciones en el orden de los mensajes.
2. El orden en que los hilos se ejecute eso depende del sistema operativo

**8) A-Join_Hilo1, A-Join_Hilo2, A-return(0)**

Esta es la situación ideal, ya que el programa será completamente sincronizado y predecible, ya que el hilo principal(main) garantiza que ambos hilos(h1 y h2) terminen antes de continuar con la ejecución.

Resultados de la prueba:

| | | | | | |
|---|---|---|---|---|---|
| **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **1:5**<br>**2:4**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **1:8**<br>**2:5**<br>**0:5** |
| **2:4**<br>**1:5**<br>**0:5** | **1:8**<br>**2:5**<br>**0:5** | **2:5**<br>**1:8**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** | **2:4**<br>**1:5**<br>**0:5** |

**Explicación.**

El código tiene sincronización ya que activar pthread_join del hilo 1 e hilo 2 garantiza que los hilos se completen su ejecución antes que el hilo principal continúe. Esto asegura que ambos hilos terminen antes de que el valor final de x se imprime 0 : ... .

El orden en que los hilos se ejecuten depende del sistema a quien le da privilegio de ejecución.

Independiente del orden en que se ejecuten los hilos, el valor final de x es siempre el resultado de las operaciones realizadas por ambos hilos.

**Caso común: 2: 4, 1: 5, O: 5**

1. El hilo 2 se ejecuta primero:
   - Incrementa i de 2 a 3.
   - Modifica x: x = 7 - 3 = 4.
   - Imprime: 2: 4.
2. El hilo 1 se ejecuta después:
   - Reinicia i a 1.
   - Modifica x: x = 4 + 1 = 5.
   - Imprime: 1: 5.
3. El hilo principal imprime:
   - 0: 5

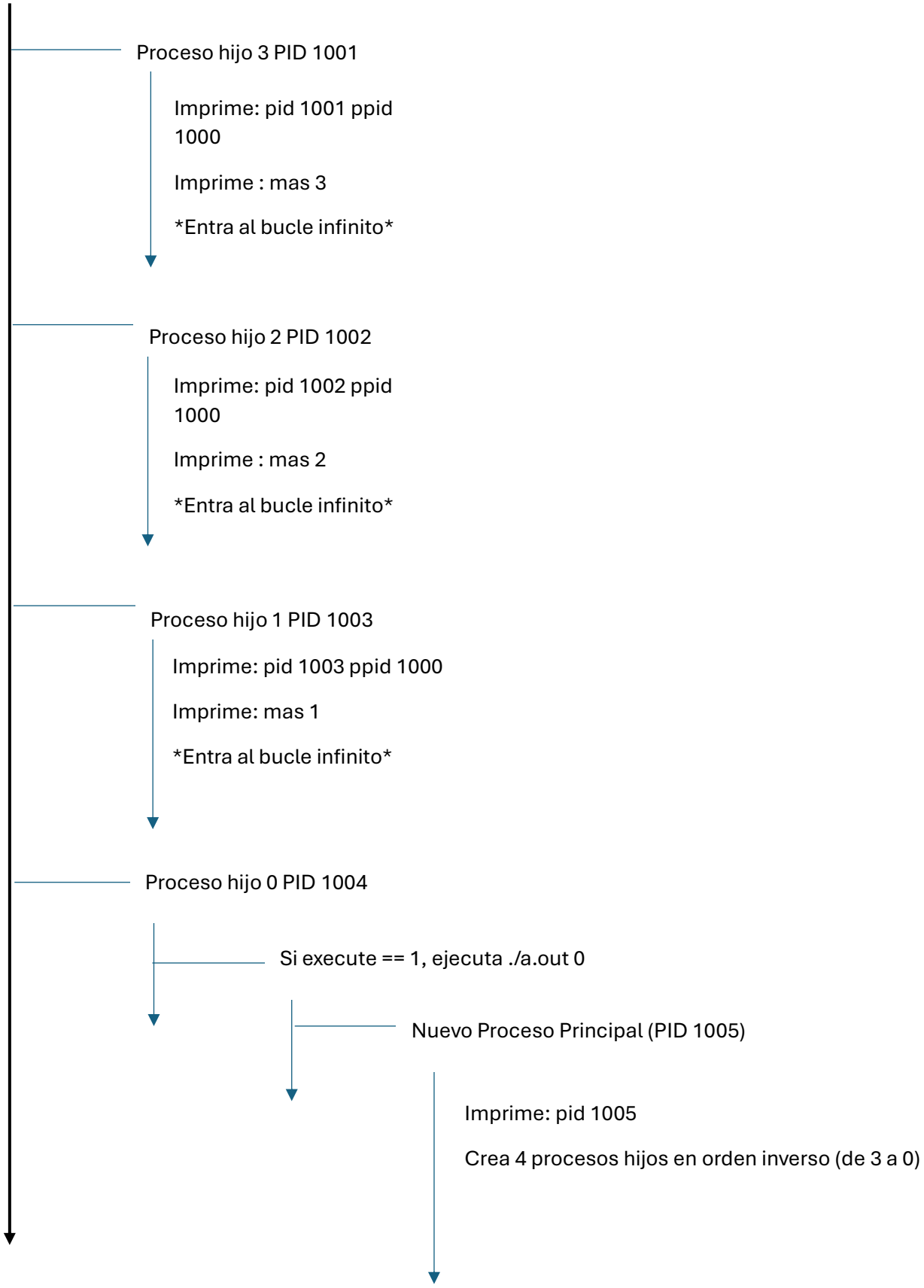
**Caso menos frecuente: 1: 8, 2: 5, O: 5**

1. El hilo 1 se ejecuta primero:
   - Reinicia i a 1.
   - Modifica x: x = 7 + 1 = 8.
   - Imprime: 1: 8.
2. El hilo 2 se ejecuta después:
   - Incrementa i de 1 a 3.
   - Modifica x: x = 8 - 3 = 5.
   - Imprime: 2: 5.
3. El hilo principal imprime:
   - 0: 5

**Caso raro: 2: 5, 1: 8, O: 5**

1. El hilo 2 modifica x primero, pero el hilo 1 sobrescribe el cambio:
   - h2: x = 7 - 3 = 4.
   - h1: x = 4 + 1 = 5.

# Ejercicio 2


Proceso hijo 3 PID 1001
Imprime: pid 1001 ppid 1000
Imprime : mas 3
*Entra al bucle infinito*
Proceso hijo 2 PID 1002
Imprime: pid 1002 ppid 1000
Imprime : mas 2
*Entra al bucle infinito*
Proceso hijo 1 PID 1003
Imprime: pid 1003 ppid 1000
Imprime: mas 1
*Entra al bucle infinito*
Proceso hijo 0 PID 1004
Si execute == 1, ejecuta ./a.out 0
Nuevo Proceso Principal (PID 1005)
Imprime: pid 1005
Crea 4 procesos hijos en orden inverso (de 3 a 0)

# Ejercicio 3

| |
|---|
| **Caso hipotético Nro. 1**<br>Hola desde aquí<br>[0]: Hola desde izquierda (cnt=1)<br>[1]: Hola desde centro (cnt=2)<br>[2]: Hola desde derecha (cnt=3)<br>Hola desde allá |
| **Caso hipotético Nro. 2**<br>Hola desde aquí<br>[1]: Hola desde centro (cnt=1)<br>[0]: Hola desde la izquierda (cnt=2)<br>[2]: Hola desde derecha (cnt=3)<br>Hola desde allá |
| **Caso hipotético Nro. 3**<br>Hola desde aquí<br>[2]: Hola desde derecha (cnt=1)<br>[0]: Hola desde izquierda (cnt=2)<br>[1]: Hola desde centro (cnt=3)<br>Hola desde allá |
| **Caso hipotético Nro. 4**<br>Hola desde aquí<br>[0]: Hola desde izquierda (cnt=1)<br>[2]: Hola desde derecha (cnt=2)<br>[1]: Hola desde centro (cnt=3)<br>Hola desde allá |

| |
|---|
| **Caso No Posible Nro. 1**<br>Hola desde aquí<br>[2]: Hola desde la derecha (cnt=3)<br>Hola desde allá<br><br>Esto se debe que si se ejecuta solamente "[2]: Hola desde la derecha (cnt=3)", se tiene cnt = 3 lo que implica que los tros hilos(i=0) y(i=1) ya se ejecutaron antes de i=2 |
| **Caso No posible Nro. 2**<br>Hola desde aquí<br>[3]: Hola desde derecha (cnt=1)<br>Hola desde allá<br><br>El hilo principal esta bloqueado en pthread_join(tid, NULL) hasta que el hilo i=2 termine complemente.<br>El hilo principal no puede ejecutar "Hola desde allá" antes que el mensaje de que el mensaje completo del hilo i = 2 sea imprimido. |
| **Caso No Posible Nro. 3** |

- [0]: Hola desde izquierda (cnt=1)
- [1]: Hola desde centro (cnt=1)
- [2]: Hola desde derecha (cnt=3)

Esto en teoría nunca se puede dar debido a que cnt a la hora de imprimir aumenta siempre de manera progresiva, cnt no se incrementa en una línea anterior, sino que aumenta en la misma línea de printf del hilo lo que obliga que siempre aumente de 1 a 3.

**OJO Este escenario si es posible tericamente ya que primero se determina el valor de myid, ptr[myid], ++cnt y el incremento de cnt no es una sola una sola instrucción, en realidad son 3 instrucciones para incrementarlo, así que sistema puede darle prioridad a otro hilo justo en ese momento.**

**Caso No posible Nro**. 4
Hola desde aquí
Hola desde allá

Este escenario nuca será posible debido a que el programa esta obligado a esperar al hilo tip **pthread_join(tid,NULL) siempre esperara al hilo con i=2, asi que en teria al menos espera un mensaje.**

# Ejercicio Nro. 4

```
/* Variables globales */
int readcnt = 0; /* Inicializamos la cantidad de escritores a 0 */
sem_t mutex, w, z; /* varibales tipo semaforos mutex and w inicializadas = 1, z = 1 */
//mutex protege el acceso exclusivo a readcnt para evitar condiciones de carrera.
//w controla el acceso a la seccion critica para los escritores(y también los lectores si no hay escritores)
//z introducido para bloquear a los lectores mientras hay escritores esperando o escribiendo, asegurando prioridad a los escritores.

void reader(void) {
  while (1) {
    P(&z);        /* Prevenir acceso si hay escritores esperando */
    //asegura que los lectores no accedan si hay un escritor esperando.
    //Si no hay escritores, el lector continúa.

    P(&mutex);    /* Acceso exclusivo a readcnt */
    //para garantizar que solo un lector acceda a la variable a la vez.
    readcnt++;
    if (readcnt == 1) {
      P(&w);      /* Bloquear escritores si es el primer lector */
    }
```

```
    V(&mutex);     /* Liberar acceso a readcnt */
    V(&z);         /* Permitir nuevos lectores si no hay escritores o escritores*/

    /* Critical section */
    /* Reading happens */
    //Decrementar el contador de lectores (readcnt):
    P(&mutex);     /* Acceso exclusivo a readcnt */
    readcnt--;     //decrementa a 0
    if (readcnt == 0) {
      V(&w);     /*  desbloquea w permitiendo escritores accder */
    }
    V(&mutex);     /* Liberar acceso a readcnt */
  }
}

void writer(void) {
  while (1) {
    P(&z);         /* Prevenir nuevos lectores mientras escribe */
    P(&w);         /* Acceso exclusivo para escribir */

    /* Critical section */
    /* Writing happens */

    V(&w);         /* Liberar acceso para lectores o escritores */
    V(&z);         /* Permitir nuevos lectores o escritores */
  }
}
```