Práctica 3

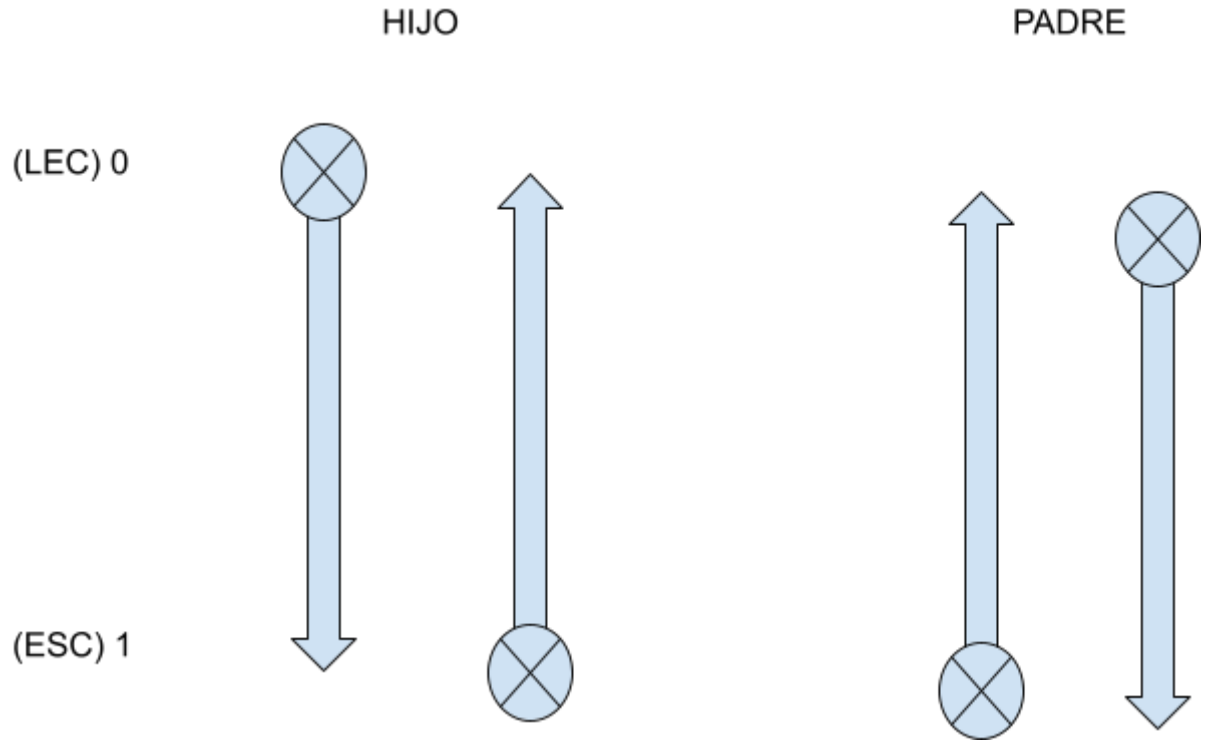
Carlos Bermúdez Expósito y Diego González Sanz

Introducción: En está práctica vamos a ver información sobre él funcionamiento de threads y mutex, además de sincronizar threads, que sean concurrentes y paralelos.

Objetivos: Él objetivo de Carlos y Diego es él perfecto entendimiento del funcionamiento de los threads y mutex, la idónea utilización, cuando utilizar concurrencias o sincronizaciones

1. Implementar un programa productor-consumidor mediante tuberías/pipes donde él proceso hijo genera un mensaje y él proceso padre los consume e imprime. Justificar el proceso 3 PUNTOS.

En este ejercicio queremos usar un full-duplex para comunicar padre e hijo, para ello nos creamos 2 pipes, uno de escritura y otro de lectura, entonces para el hijo queremos cerrar el lado de lectura en un pipe que en este caso es “p” y el de escritura en el pipe “p2”, para que así el hijo escriba mediante la función write en el buffer y por el contrario hacer totalmente lo opuesto en el padre para que lea en vez de escribir, cerramos el lado de escribir del pipe “p” (1) y cerramos el lado de leer del “p2”. Usamos un full-duplex aunque se podría hacer solo con uno por la posible de gestión de errores.



EJERCICIO_2:
Para este ejercicio queremos crearnos un bound buffer, para ello queremos cubrir además los siguientes puntos:

1. El proceso productor produce información y la almacena en un buffer.
2. El proceso consumidor accede al buffer y consume la información.
3. El productor y el consumidor comparten variables.
4. El productor no puede acceder al buffer si está lleno.
5. El consumidor no puede acceder al buffer si está vacío.

Entonces lo primero nos creamos 2 semáforos uno para el producto y otro para el consumidor.

1-El proceso productor comprueba si el buffer está vacío para meterle información, luego tenemos que saltar una posición pra que los 2 punteros no apunten a lo mismo y se cree un bucle infinito. Todo esto almacenado en buffer ya que es una variable global y puede acceder a ella. Además para acceder al buffer tiene un sleep random para escribir en el buffer y no haya problemas con el consumidor y se haga de manera secuencial.

2-El proceso consumidor accede al buffer de la misma manera que el productor, porque es global y consume la información si no es =0, es decir si no está vacío, para que no nos de un error. De la misma manera el consumidor tiene su sleep para la misma función que el productor.

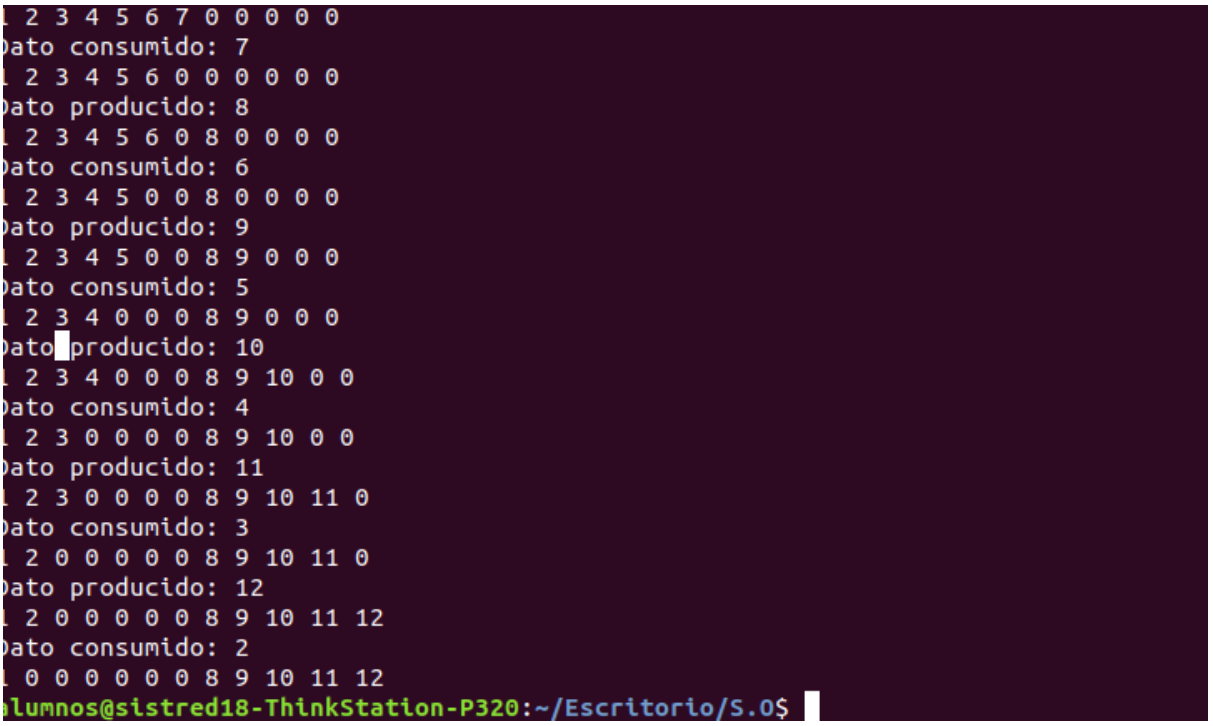
3-Comparten variables tales como un array de, en este caso, 12 posiciones llamado buffer que es una variable global a la cual puede acceder el consumidor y el productor. Y la utilización de los semáforos para su correcto funcionamiento.

4 Además hemos hecho una condición para saber cuando no está completo que pueda meter información en él buffer sin problema con la siguiente linea de codigo (‘if (buffer[i] == 0)’), eso significa que si en esa posicion e igual a 0 hay espacio libre.

5 Además hemos hecho una condición para saber cuando hay algo de información que pueda acceder, lo hemos hecho con la siguiente línea de código ('if (buffer[i] != 0) '), eso significa que si en esa posición es distinto a 0 es que hay un valor.

Ademas hemos hecho distintas pruebas con los sleep para ver su funcionamiento, este es el ejemplo del random pero tambien lo hemos hecho con un sleep 2 y 20, llegando a la

conclusion que se llena y luego se va borrando poco a poco:



Conclusión: La conclusión que hemos sacado es que en está práctica hemos aprendido a utilizar semáforos, además de hacer uso de pipes (tuberías) de una y full- duplex las cuales pueden usarse como prevención de posibles errores y un buffer circular con sus productores y consumidores llegando a algunas conclusiones como que si los 2 punteros apuntan al mismo sitio se podría desencadenar un bucle infinito.