Actividad Unidad didáctica 2

Exclusión mutua y sincronización

Alumno Javier Raneros Cartujo

Asignatura Diseño de Sistemas Operativos

Grado en Ingeniería Informática

curso 2025-2026

Universidad Isabel I

Índice

1. Introducción

En esta actividad es necesario crear una solución para el problema del productor/consumidor utilizando Java. En este problema tenemos uno o varios threads que producen datos a un ritmo diferente. Estos datos se almacenan en un buffer que será consumido a un ritmo diferente por cada thread.

Requisitos de la implementación:

- Utiliza wait and notify para implementar esta solución.
- Cuando arranque el programa se preguntará al usuario cuántos productores utilizar
- Cuando arranque el programa se preguntará al usuario cuántos consumidores utilizar

2. Desarrollo y solución

El problema propuesto precisa que la actividad a entregar ponga en ejecución varios hilos de forma que trabajen de forma concurrente, dentro de lo que sería una **aplicación estructurada concurrente**, en la que los hilos productores y consumidores, trabajarán de forma sincronizada en los procesos de producir y consumir productos (en este caso enteros) almacenados en un buffer común compartido y punto crítico de la sincronización.

Como solución y puesto que no se indica como requisito, el buffer compartido se comportará como una pila o lista LIFO ya que para la sincronización de los hilos productores / consumidores no es significativo el modo en que sean introducidos y extraídos del buffer los elementos, sino la propia sincronización para evitar desbordamiento y/o accesos indebidos.

El buffer será instanciado al inicio de la aplicación y compartido con todos los hilos de forma concurrente durante la ejecución del proceso principal, los métodos ofrecidos a las clases de producción y consumición se declararán como síncronos e informará por consola del avance de los productos y del estado de ocupación de la pila. El buffer es por tanto el centro de toda la concurrencia que creará los monitores para cada hilo que ejecute sus métodos síncronos para que pasen a estado de espera (*wait()*) o avisarles que podrían volver a estado de ejecución (*notify()*).

Para realizar la actividad en Java se ha separado el problema en las siguientes clases princiaples:

- AppProductorConsumidor (Main)
- Productor Thread
- Consumidor Thread
- Buffer Pila (monitores wait() y notify())

Además de todo ello también se ha incluido una clase Logger con la que extraer en un log las tareas realizadas y examinar posteriormente el trabajo de todos los hilos.

AppProductorConsumidor

Esta clase es la propia aplicación que ejecuta todo donde se encuentra el main. Como uno de los requisitos principales al ejecutarse pide por consola el número de hilos productores y consumidores que se deben crear, no permitiendo introducir valores inferiores o iguales a cero,

ya que en caso de solo existir un tipo de hilo, sería imposible la sincronización, ya que en caso de solo existir de un tipo, el otro tipo que se haya creado sufrirá un bloqueo completo durante la ejecución.

Desde esta clase se configura entre otras cosas importantes (no se ha diseñado ninguna opción de propiedades o petición por consola para hacer cambios de estos datos):

- **CAPACIDAD_BUFFER**: Será el número máximo de elementos que manejará el buffer en memoria, por defecto se ha configurado con 50 elementos como tamaño máximo.
- **Ritmo Productor y Ritmo Consumidor**: para parametrizarlos se utilizan las variables MIN_RITMO_PRODUCTOR, MAX_RITMO_PRODUCTOR, MIN_RITMO_CONSUMIDOR, MAX_RITMO_CONSUMIDOR para que la aplicación calcule de forma aleatoria valores entre esos márgenes mínimo y máximo respectivamente. Por defecto se configura para que los valores aleatorios obtenidos sean entre 500 y 900 milisegundos.

Desde esta clase una vez solicitados el número de hilos productores y consumidores a crear, se crea el Buffer en memoria que será inyectado en todos los hilos que se creen.

Se crearán tantos hilos como productores / consumidores como se hayan indicado por consola y cada hilo será creado con un ritmo diferente de forma aleatoria dentro de los parámetros establecidos e inyectado con el Buffer creado mediante el constructor de las clases productoras y consumidoras.

Por último se inician mediante el método start() todos hilos creados.

Productor / Consumidor

Las clases Productor y Consumidor extienden de la clase Thread para implementar la sincronización de los hilos.

Las dos clases en su método constructor crean con los objetos con un identificador para hacer un seguimiento de qué hilos con qué frecuencia y elementos han producido y/o consumido.

En este método constructor también se define el **ritmo** con el que va a trabajar, que sencillamente lo que va realizar es una pausa o sleep de los milisegundos que se pasen por parámetro.

Y lo más importante en el constructor también se **inyecta el Buffer** que será accedido de forma compartida por todos los hilos creados, es el mismo objeto al que accederán todos los hilos.

Las dos clases sobrescriben el método run() que será el ejecutado en el momento que cada hilo se inicie mediante su start():

- **Productores**: En el caso de los hilos productores producirán un elemento que en nuestro caso es un entero secuencial y será producido/insertado en el buffer mediante el método buffer.ejecutaProducir()
- **Consumidores**: En el caso de los consumidores ejecutarán el método buffer.ejecutaConsumir() que devolverá o consumirá del buffer el elemento correspondiente a salir de la pila.

La ejecución del método run() por parte de los productores/consumidores se ha desarrollado para no tener ningún límite durante la ejecución, se ejecutará hasta el infinito y el programa deberá ser finalizado desde la consola.

Importante resaltar que es en este método run() dónde se gestiona el ritmo del productor/consumidor según lo establecido durante su creación.

Buffer

La clase Buffer se podría considerar que es la parte más importante de todas, porque es el recurso compartido que debe sincronizar todos los hilos para que los accesos a la pila se realicen correctamente y en orden, sincronizando a productores y consumidores para que accedan correctamente a la pila.

Como se ha comentado, el buffer gestiona un array de enteros en modo pila o lista LIFO que cuando es instanciada se pasará por parámetro el número máximos de elementos a almacenar, que por defecto será de 50 elementos.

Para gestionar su producción / consumición tiene los siguientes métodos que se ejecutarán en modo sincronizado utilizando los monitores **wait**() y **notifify**():

- **synchronized void ejecutaProducir**(int numeroProducido): Método que será el invocado por los hilos productores desde su método run()
- **synchronized int ejecutaConsumir**(): Método que será el invocado por los hilos consumidores desde su método run()

Y es en estos métodos donde los hilos productores irán apilando los elementos producidos en la pila y los hilos productores irán extrayendo los datos de la pila de forma concurrente y sincronizada, teniendo en cuenta que para conseguir sincronizar todos los hilos mediante los monitores **wait()** y **notifiy()**:

- Al producir mediante ejecutaProducir() si la capacidad del buffer ha llegado al máximo, el hilo entrará en modo espera mediante el método wait() de los hilos.
- Al consumir mediante ejecutaConsumir() si la el buffer se encuentra vacio, el hilo consumidor entrará en modo espera igualmente.
- Cada vez que o bien un hilo productor o un hilo consumidor hace su trabajo, notifica mediante notify() al hilo que se haya quedado en espera de ese monitor, para que salga del modo espera y continúe su ejecución.

3. Ejecución y pruebas

Como se ha indicado anteriormente, la clase que tiene el método main() a ejecutar es la clase **AppProductorConsumidor**. Para ver cómo trabaja la aplicación se ha añadido al Buffer un método que nos permitirá ver de forma más gráfica la evolución de los hilos produciendo y consumiendo y el propio estado del buffer de forma gráfica.

Por defecto como se ha indicado los parámetros de configuración de los hilos serán con los siguientes datos para las pruebas que vamos a realizar:

- Tamaño del buffer: 50. Se ha comprobado que para hacer pruebas medianamente significativas es conveniente que tengan un tamaño amplio de entre 25 y 100 elementos al menos.

- Milisegundos generados aleatoriamente entre 500 y 900 de pausa que marcarán el ritmo de los hilos productores / consumidores.

Las pruebas a realizar para ver en funcionamiento los hilos serán:

- Igual número de hilos productores / consumidores
- Número hilos Productores > Consumidores
- Número hilos Productores < Consumidores

De igual forma también es conveniente que el número de hilos sea significativo para ver en funcionamiento de la sincronización en el buffer.

Igual número de hilos productores y consumidores

Se crean 20 hilos de cada tipo con los valores por defecto. Los resultados son los siguientes:

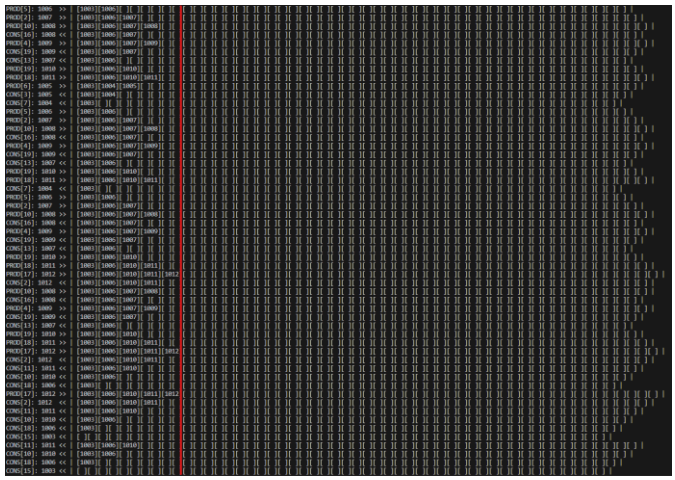
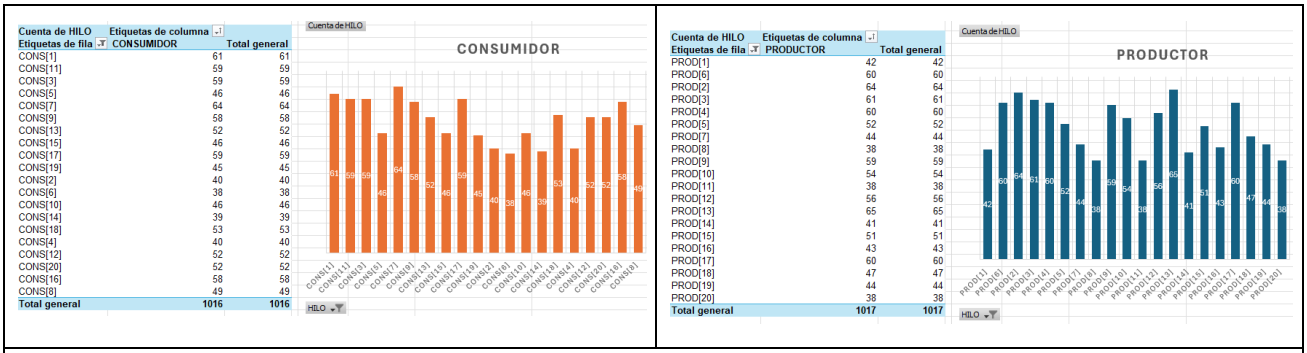


Figura 1. Ejecución de mismo número de hilos productores que de consumidores

En esta prueba se aprecia que, en igualdad de condiciones, el buffer se mantiene estable, llegando casi a vaciarse y nunca superando el 25% de la capacidad máxima.

Según el log las interacciones de los hilos son al 50%, con lo que se comprueba que el sistema operativo ha gestionado equitativamente los hilos y se cumple que a igual número de hilos y ritmos similares, se mantiene la equidad en el reparto de tiempo de ejecución.



Cuenta de HILO	Etiquetas de columna	
Etiquetas de fila	CONSUMIDOR	Total general
CONS[1]	61	61
CONS[11]	59	59
CONS[3]	59	59
CONS[5]	46	46
CONS[7]	64	64
CONS[9]	58	58
CONS[13]	52	52
CONS[15]	46	46
CONS[17]	59	59
CONS[19]	45	45
CONS[2]	40	40
CONS[6]	38	38
CONS[10]	46	46
CONS[14]	39	39
CONS[18]	53	53
CONS[4]	40	40
CONS[12]	52	52
CONS[20]	52	52
CONS[16]	58	58
CONS[8]	49	49
Total general	1016	1016

Cuenta de HILO	Etiquetas de columna	
Etiquetas de fila	PRODUCTOR	Total general
PROD[1]	42	42
PROD[6]	60	60
PROD[2]	64	64
PROD[3]	61	61
PROD[4]	60	60
PROD[5]	52	52
PROD[7]	44	44
PROD[8]	38	38
PROD[9]	59	59
PROD[10]	54	54
PROD[11]	38	38
PROD[12]	56	56
PROD[13]	65	65
PROD[14]	41	41
PROD[15]	51	51
PROD[16]	43	43
PROD[17]	60	60
PROD[18]	47	47
PROD[19]	44	44
PROD[20]	38	38
Total general	1017	1017

Figura 2. Interacción hilos consumidores y productores en una ejecución con el mismo número de hilos

Se demuestra que la sincronización cumple su cometido porque el buffer ni se desborda ni se queda bloqueado vacío.

Número hilos Productores > Consumidores

Se crean 20 hilos productores y 19 hilos consumidores con los valores por defecto. Los resultados son los siguientes:

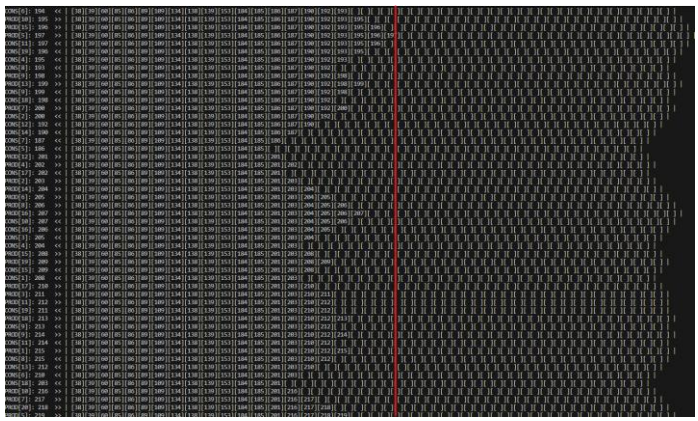
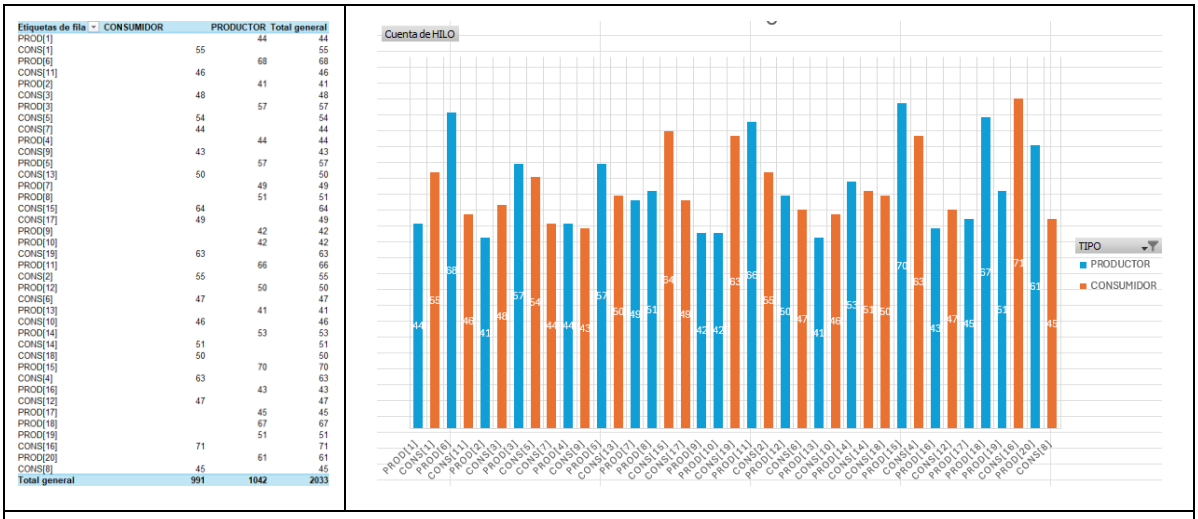


Figura 3. Ejecución con mayor numero de hilos productores que consumidores. El proceso de llenado de la pila continua gradualmente hasta llenarse

Se aprecia que en el momento de que hay un hilo más por parte de la producción, el buffer parece que se mantiene estable, pero poco a poco comienza a llenarse finalmente hasta la totalidad, pero de tal modo que siempre algún hilo consumidor puede extraer una o dos unidades.

Las interacciones de los hilos según el log demuestran que los productores son superiores en un 51,25%, con lo que se verifica que el sistema operativo ha gestionado de nuevo correctamente los hilos y cumple que a mayor número de hilos productores, al aumentar la tasa de producción, el buffer se llena gradualmente.



Etiquetas de fila	CONSUMIDOR	PRODUCTOR	Total general
PROD[1]		44	44
CONS[1]	55		55
PROD[6]		68	68
CONS[11]	46		46
PROD[2]		41	41
CONS[3]	48		48
PROD[3]		57	57
CONS[5]	54		54
CONS[7]	44		44
PROD[4]		44	44
CONS[9]	43		43
PROD[5]		57	57
CONS[13]	50		50
PROD[7]		49	49
PROD[8]		51	51
CONS[15]	64		64
CONS[17]	49		49
PROD[9]		42	42
PROD[10]		42	42
CONS[19]	63		63
PROD[11]		66	66
CONS[2]	55		55
PROD[12]		50	50
CONS[6]	47		47
PROD[13]		41	41
CONS[10]	46		46
PROD[14]		53	53
CONS[14]	51		51
CONS[18]	50		50
PROD[15]		70	70
CONS[4]	63		63
PROD[16]		43	43
CONS[12]	47		47
PROD[17]		45	45
PROD[18]		67	67
PROD[19]		51	51
CONS[16]	71		71
PROD[20]		61	61
CONS[8]	45		45
Total general	991	1042	2033

Figura 4. Interacción de hilos en una ejecución con mayor número de hilos **productores**

Se demuestra nuevamente que la sincronización cumple su cometido porque el buffer aún llenándose, no desborda la pila y siguen produciendo y consumiendo la aplicación.

Número hilos Productores < Consumidores

Se crean 19 hilos productores y 20 hilos consumidores con los valores por defecto. Los resultados son los siguientes:

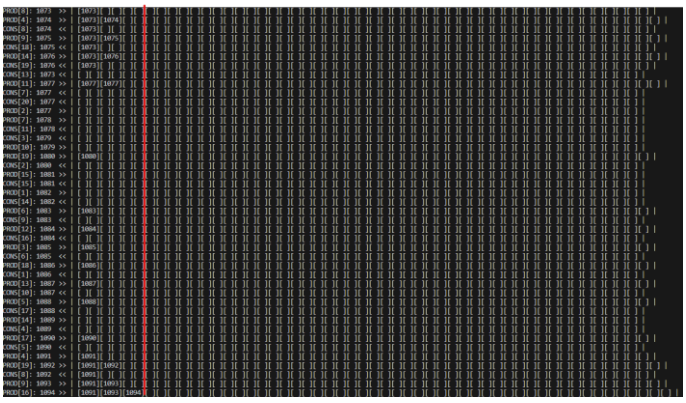


Figura 5. Ejecución mayor número de hilos **consumidores**. El buffer se mantiene casi vacío

De nuevo se aprecia que en el momento de que hay un hilo más en este caso de un consumidor, el buffer llega a vaciarse por momentos, pero de tal modo que siempre algún hilo productor puede apilar una unidad.

Las interacciones de los hilos según el log no demuestran que los consumidores sean superiores porque se mantienen al 50%, y esto es porque llegado un momento que no hay más producciones, los consumidores tampoco consumen más porque no existe el producto, con lo que se verifica que el sistema operativo ha gestionado de nuevo correctamente los hilos y cumple que a mayor número de hilos consumidores llega un punto que la tasa de producción y consumo se iguala.

Etiquetas de fila	CONSUMIDOR	PRODUCTOR	Total general
CONS[1]	57		57
CONS[10]	53		53
CONS[11]	63		63
CONS[12]	44		44
CONS[13]	52		52
CONS[14]	47		47
CONS[15]	55		55
CONS[16]	66		66
CONS[17]	70		70
CONS[18]	50		50
CONS[19]	47		47
CONS[2]	66		66
CONS[20]	48		48
CONS[3]	53		53
CONS[4]	58		58
CONS[5]	44		44
CONS[6]	69		69
CONS[7]	51		51
CONS[8]	49		49
CONS[9]	50		50
PROD[1]		47	47
PROD[10]		62	62
PROD[11]		51	51
PROD[12]		59	59
PROD[13]		71	71
PROD[14]		69	69
PROD[15]		55	55
PROD[16]		44	44
PROD[17]		50	50
PROD[18]		56	56
PROD[19]		77	77
PROD[2]		46	46
PROD[3]		69	69
PROD[4]		52	52
PROD[5]		58	58
PROD[6]		72	72
PROD[7]		54	54
PROD[8]		44	44
PROD[9]		56	56
Total general	1092	1092	2184

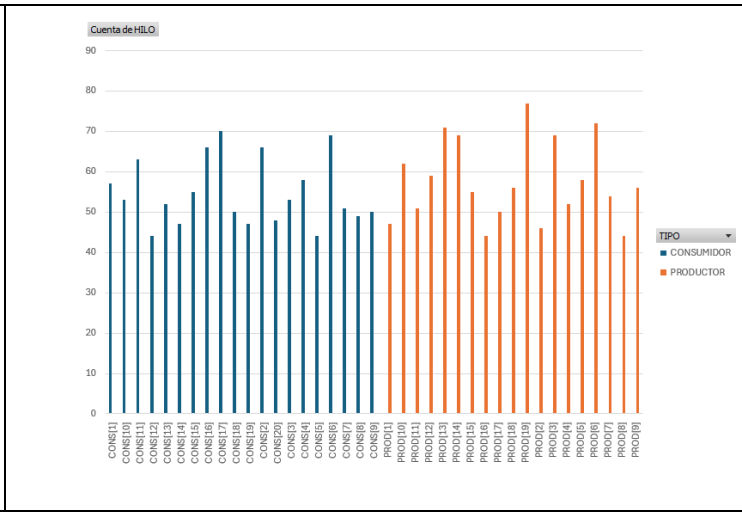


Figura 6. Interacción de hilos en una ejecución con mayor número de hilos **consumidores**

Se demuestra también que la sincronización cumple su cometido porque el buffer aun vaciándose por completo no se bloquea y continúa admitiendo elementos producidos.

4. Conclusiones

Aún estableciendo para las pruebas que se inicien unos hilos u otros para que no tengan ventaja unos hilos sobre otros en el arranque de la aplicación, sobre todo si se crean muchos hilos, se concluye que el resultado final no influye o no determinante el resultado final la sincronización.

Por todo ello y con las pruebas realizadas, se concluye que en igualdad de hilos, el sistema operativo es equitativo y el sistema permanece estable, pero en el momento en que se desequilibra la el número hilos a ejecutar por parte de alguno, aunque sea un desequilibrio pequeño, al aumentar la tasa de consumo o la tasa de producción, la pila acaba llenándose o vaciándose según sea el desequilibrio, pero en ningún momento se produce ningún error o condición de carrera que pueda provocar bloqueos o accesos que deriven en un desbordamiento.

Proyecto Java en GitHub: https://github.com/javierraneros-cell/ui1_dis_sop_ud2

5. Bibliografía

[1] Java® Platform, Standard Edition & Java Development 23 API Specification, «notify», Accedido: 27 de marzo de 2026. [En línea]. Disponible en: https://download.java.net/java/early_access/valhalla/docs/api/java.base/java/lang/Object.html#notify()