Ejercicios de Concurrencia (Threads)

1. Implemente un programa que escriba un "hola mundo" por cada hilo de ejecución que se cree (seis es un número razonable) y que además indique desde que hilo se imprime. Luego haga que cada uno espere un tiempo proporcional a su identificador antes de imprimir el mensaje (el thread 1, un segundo, el 2, dos segundos, el 3, tres segundos,…). Lance los threads mediante la clase Thread y luego mediante el interfaz Runnable ¿Qué diferencias aprecia?

2. Implemente un programa que lance cuatro threads, cada uno incrementará una variable contador de tipo entero, compartida por todos, 5000 veces y luego saldrá. No se preocupe de sincronizar los accesos a dicha variable. ¿Obtiene el resultado correcto? Ahora sincronice el acceso a dicha variable. Lance los threads primero mediante la clase Thread y luego mediante el interfaz Runnable. Haga una última versión con AtomicInteger.

3. Implemente un programa que reciba a través de sus argumentos una lista de ficheros y cuente cuantas líneas, palabras y caracteres hay en cada fichero, así como el total entre todos los ficheros (el 'wc' de linux). Modifíquelo para que se cree un thread por cada fichero a contar, cuidando de obtener igualmente el total. Mida cuanto tiempo emplea la versión secuencial para contar unos cuantos ficheros grandes y cuanto tarda la versión paralela para hacer lo mismo.

4. Implemente una simulación de la fabula que cuenta la carrera entre la liebre y la tortuga. Para hacerlo más interesante la carrera será cuesta arriba por una pista resbaladiza, de modo que a veces podrán resbalar y retroceder algunas posiciones. Habrá un thread que implementará la tortuga y otro la liebre. Cada uno se suspenderá durante un segundo y luego evaluará lo que ha pasado según unas probabilidades:

Animal	**Suceso**	**Probabilidad**	**Movimiento**
Tortuga	Avance rápido	50%	3 casillas a la derecha
	Resbaló	20%	6 casillas a la izquierda
	Avance lento	30%	1 casilla a la derecha
Liebre	Duerme	20%	
	Gran salto	20%	9 casillas a la derecha
	Resbalón grande	10%	12 casillas a la izquierda
	Pequeño salto	30%	1 casilla a la derecha
	Resbalón pequeño	20%	2 casillas a la izquierda

   Calcule la probabilidad con random, de 1 a 100 y determine con dicho número que ha hecho cada animal. Considere que hay 70 casillas, de la 1 a la 70, la 1 de salida y la 70 de llegada. Si resbala al principio vuelve a la 1, nunca por debajo. Tras cada segundo y después de calcular su nueva posición imprima una línea por cada animal, con blancos de 1 la posición-1 y luego una letra T para la tortuga y una L para la liebre. Imprima al comienza de la carrera un mensaje. Después de imprimir las líneas determine si alguno ha llegado a meta y ha ganado, imprimiendo un mensaje. Si ambos llegan a la vez declare un empate.

5. Implemente un programa secuencial que calcule el producto de dos grandes matrices. Después modifíquelo para que esta tarea se realice entre cuatro threads, cada uno ocupado de un subconjunto de la matriz resultado. Mida el tiempo que emplea cada una de las versiones.

6. Implemente una barrera. Una barrera es un punto de sincronización entre varios threads. Se caracteriza porque los threads que van llegando a ella esperan hasta que llega el último. Por ejemplo, se puede sincronizar N threads y hasta que el último no llegue los demás no deben poder continuar. Pruébela con el código del primer problema, es decir lancé unos pocos threads (cuatro es razonable), cada uno que espere un tiempo proporcional a su

identificador, que luego imprima un mensaje que lo identifique y que luego espere en la barrera por los otros threads. Introduzca este esquema en un bucle para ver como evoluciona con el tiempo.

7. Implemente un esquema productor/consumidor. Es decir, habrá un thread que se dedicará a crear datos y otro a consumirlos. Es un esquema bastante habitual de concurrencia, con uno o varios threads creando datos y con uno o varios consumiendo dichos datos. En este caso habrá sólo uno de cada. Además, por simplicidad, los datos a producir/consumir serán enteros, empezando por el 1, luego el 2, el 3 y así sucesivamente hasta el 30. Cada thread esperará un tiempo aleatorio entre datos (random), entre 0 y 2 segundos, para simular el tiempo que tarda en producir/consumir cada dato. Utilice un buffer circular, con capacidad para 4 datos, para pasar los datos entre los dos threads. Asegúrese de sincronizarlos correctamente (wait/notify) para que el productor no meta datos en el buffer cuando este esté lleno ni que el consumidor coja datos cuando este está vacío. Haga que el consumidor imprima los datos una vez que los haya consumido (tras la espera) para ver que no se pierde ninguno. Imprima también un mensaje cada vez que un thread tenga que esperar porque el buffer esté lleno o vacío.