**1. Desarrolla aplicaciones compuestas por varios procesos reconociendo y aplicando principios de programación paralela.**

Criterios de evaluación:

**a) Se han reconocido las características de la programación concurrente y sus ámbitos de aplicación.**

Hemos hechos prácticas sobre workers y buckets o bloques. En estas prácticas, lo que buscamos es partir en bloques y ejecutar de forma paralelas diferentes hilos de ejecución. De esta manera tenemos un programa que puede aprovechar mejor los recursos que tiene nuestro ordenador, haciendo que varios núcleos trabajen a la vez, cada uno cogiendo un hilo de ejecución diferente.

Para hacerlo más visual, hemos usado un html y JavaScript que nos permite aplicar efectos a imágenes utilizando un lienzo.

Primero lo hemos hecho directamente, cambiando la imagen completa del tirón usando un worker al que le envío unos datos y me devuelve una respuesta.

Después, nos hemos centrado en dividir la imagen en diferentes bloques en función de los pixeles. De esta forma, hemos podido cambiar uno a uno los pixeles en vez de la imagen completa y además distribuir este trabajo entre diferentes workers.

**c) Se han analizado las características de los procesos y de su ejecución por el sistema operativo.**

Mientras hacíamos las prácticas hemos visto como los diferentes núcleos del procesador iban trabajando de manera simultanea al mandar la información mediante hilos. De esta forma hemos comprobado como optimizar los recursos de nuestro ordenador, no tanto para evitar sobrecargarlo sino para sacarle el máximo rendimiento.

**d) Se han caracterizado los hilos de ejecución y descrito su relación con los procesos.**

Uno de los principales objetivos es maximizar el rendimiento y la eficiencia nuestro ordenador.

Es importante saber que los ordenadores pueden tener varios procesadores y cada uno de ellos puede manejar múltiples hilos y procesos. Los hilos van a ejecutarse de manera concurrente permitiendo varias tareas simultaneas dentro de la misma aplicación.

Al dividir una tarea grande en subtareas más pequeñas que pueden ejecutarse al mismo tiempo, se reduce significativamente el tiempo total de procesamiento y se hace un uso más eficiente de la CPU, la memoria y otros recursos del sistema.

Para todo esto se han usado los workers y los buckets

**e) Se han utilizado clases para programar aplicaciones que crean subprocesos.**

Concretamente, para llevar acabo esta forma de programar, hemos usado los workers.

Desde 2010, son una característica de HTML5 que permite la ejecución de scripts en segundo plano en un navegador web, sin interferir con la ejecución de la interfaz de usuario. Esto es especialmente útil para realizar tareas intensivas en computación, ya que permite que la aplicación siga siendo receptiva mientras se realizan cálculos complejos o se procesan grandes cantidades de datos.

Al ejecutar un hilo separado del hilo principal, las tareas que consumen mucho tiempo no bloquean la interacción del usuario con la página.

Los workers no pueden recorrer el DOM tal como lo hacemos con JavaScript, por lo que no pueden manejar directamente los elementos de la interfaz de usuario. Sin embargo, existe una forma concreta de enviarle datos y que luego él los devuelva al hilo principal. Para ello, usamos una forma de comunicación asincrónica mediante el postMessage y el onmessage.

Desde un script, le enviamos un postMessage que el worker recibe. Realiza los procesamientos necesarios y cuando han terminado devuelve los datos mediante un onmessage.

**f) Se han utilizado mecanismos para compartir información con los subprocesos iniciados.**

En esta parte vamos a profundizar un poco más en como funcionan el postMessage y el onmessage.

El método postMessage se utiliza para enviar mensajes desde un contexto a otro. En el caso de workers, se usa para enviar datos desde el hilo principal (la interfaz de usuario) al worker.

En el hilo principal se establece un postMessage(data), donde data puede ser cualquier tipo de dato que se pueda serializar, como objetos, arrays, cadenas, etc.

En el worker se define un onmessage para recibir el mensaje del hilo principal.

Al recibir el mensaje, el worker procesa la información y utiliza postMessage para enviar el resultado de vuelta al hilo principal.

Ese hilo principal también tiene un onmessage para recibir las respuestas desde el worker y poder sacarla en consola.

**g) Se han utilizado mecanismos para sincronizar y obtener el valor devuelto por los subprocesos iniciados.** *Mezcla entre pasar información en postMessage y los semáforos*

Es importante, además de tener la posibilidad de hacer procesamientos en paralelo, saber cuando todos han terminado. Ya que no todos los workers van a terminar a la misma vez o no están haciendo lo mismo. Y es posible que necesitemos asegurarnos de que todos ellos hayan terminado antes de combinar los resultados, por ejemplo. Por ello podemos usar la idea de los semáforos que nos indica si todos han terminado.

Esto se hace estableciendo el número de workers que vamos a usar mediante una variable que recoja este dato. Además creamos otra variable para el semáforo.

Conforme los workers vayan enviado sus postMessage de que han terminado, la variable semáforo va incrementándose hasta se igual a la variable que recoge el número de workers. En ese momento, nos lanzará un mensaje diciendo que todos los workers han terminado y así sabremos que ya tenemos todos los datos necesarios procesados por los workers.

**h) Se han desarrollado aplicaciones que gestionen y utilicen procesos para la ejecución de varias tareas en paralelo. –**

Ver el código en gitHub donde se recogen las practicas de los workers y los buckets

**i) Se han depurado y documentado las aplicaciones desarrolladas.**

En este caso, dado que las practicas de los worker son muy cortitas cada una de ellas, las modificaciones profundas en el código se han hecho en los archivos:

014-miProcesadorDeWorkers.html y 014-miProcesadorDeWorkers.js

Aquí se ha dejado comentado todo el código de las distintas partes que se han creado para realizar el ejercicio.