PRÁCTICA 4

Programación concurrente

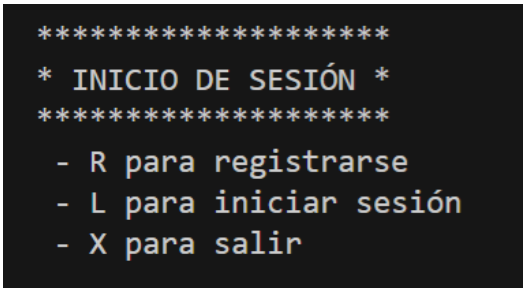
21711787
Marcos Eladio Somoza Corral

INDICE

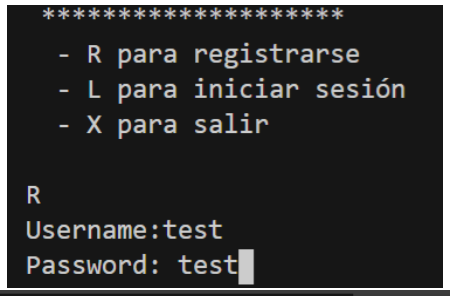
Login

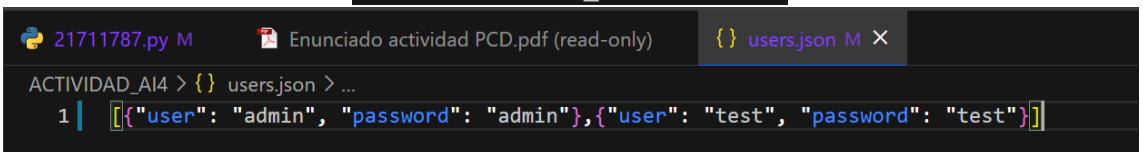
Para hacer el sistema de login se ha optado por la serialización en formato json mediante `import json`. Para ello se ha creado una clase User, encargada de guardar la información del nombre de usuario y la contraseña (además de un método para convertirlo a diccionario los datos del usuario); cabe destacar que en el constructor se añade a la lista de usuarios registrados inicializar el objeto.

Para administrar la lógica y la información de inicio de sesión, se ha creado la clase DataController, en cuyo inicio accede al path donde está guardado el json con los registros de usuarios y de él recoge todos los usuarios en una lista, también inicializa la interfaz de inicio de sesión.

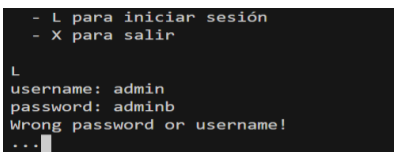


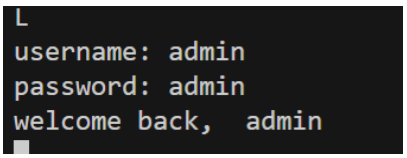
Cuando se registra un nuevo usuario y dado el scope del proyecto (no se comprueba si existe ya un usuario registrado con el mismo nombre), se crea un objeto User y se serializa al json anteriormente mencionado, también se vuelve a la pantalla de inicio de sesión.





Cuando se inicia sesión se recorre la lista de Usuarios que ha recogido del fichero json el objeto DataController cuando fue inicializado y si se da un caso en el que tanto el usuario como la contraseña sean iguales, se accede a la pantalla de ejercicios. Si por el contrario no encuentra coincidencias, vuelve a la pantalla de inicio de sesión.





Ejercicio 1

Codificar un programa que calcule el producto de dos matrices cuadradas de orden n^3 de números enteros, se deberá paralelizar el procesamiento en función del número de cores que tenga la máquina donde se ejecute el código.

Para multiplicar matrices de gran tamaño de forma concurrente y paralela, es decir, para poder utilizar simultáneamente varios cores y agilizar el tiempo de ejecución, se deben dividir las tareas en base a las filas y columnas de las matrices. Cada core iterará sobre su fila y columna asignada y hará el cálculo correspondiente A[i][k] * B[k][j] para guardarlo en la posición pertinente dada por [i*len(B[0]) + j] (dentro de la triple iteración fila columna, longitud de la fila B y de la fila A) dentro de la matriz compartida para todos los cores. Esta "matriz" compartida (más bien, un array unidimensional) se crea gracias a multiprocessing.RawArray.

Finalmente, una vez se han ejecutado todos los cores y han acabado sus cálculos, se deberá convertir de vuelta el array compartido a una matriz bidimensional, a fin de guardar los datos en una matriz resultante igual que las multiplicadas.

Cabe destacar que no se ha logrado realizar este cálculo con el número de elementos pedido (el número de expediente 21711787 en este caso) dadas las limitaciones de utilizar un solo equipo. Si se quisiera realizar un cálculo semejante, se debería tener en consideración el uso de clústers o en su defecto (y preferiblemente, dada la versatilidad y rapidez que ofrece) un servicio web como Google Cloud.

Ejercicio 2

Se pide codificar el algoritmo mergeSort para un array n, donde n es :n = vuestro número de expediente.(En el ej. Usamos el exp. 21535220) Se deberá generar el arrary n con 21.535.220 elementos de números enteros (deberán ser aleatorios para poder ordenarlos).Ej. mergeSort(5)RESULTADO = imprima el array ordenado [0,2,3,4,4]

En primera instancia, se ha creado un array de tamaño igual al número de expediente utilizado ProcessPoolExecutor. Se ha dividido el tamaño (expediente) entre el número de cores que se van a utilizar, y que cada uno de ellos cree un array con elementos n = expediente/num_cores de valor aleatorio entre 0 y expediente.

Una vez está creado el array, se aplica mergesort al mismo. Para paralelizar en base al número de procesadores de la máquina, se van asignando llamadas recursivas del algoritmo a los cores hasta que no queden más; ahí se pasa a llamar al mergesort recursivo clásico (aquel que no reparte mas llamadas recursivas a otros cores).

Es importante tener en cuenta que el array que se está ordenando es convertido a un tipo de array conocido como compartido, mediante multiprocessing.RawArray de tipo unidireccional, sin sincronización.

Ejercicio 3

Se pide codificar un algoritmo que obtenga el número n correspondiente a la secuencia de Fibonacci, donde n es : n= vuestro número de expediente.

Fibonacci tiene muchas manera de calcularse, cada cual con una complejidad O mejor o peor. Para poder dividir los cálculos entre los núcleos disponibles se ha utilizado aquel que calcula Fibonacci mediante el exponente de matrices. Este método convierte la premisa de Fibonacci a matrices, donde para calcular Fib(n) de manera recursiva usamos fib(n) = fib(n-1) + fib(n-2), ahora lo aplicamos en forma de matrices.

Al tomar esa igualdad de fib(n) como cierta, podemos afirmar también que fib(n+1) = fib(n) + fib(n-1)

Si vectorizamos f(n) y f(n-1) en v tal que v = { f(n) , f(n-1) } siendo v el anterior número de Fibonacci para este caso. Si también vectorizamos f(n+1) y f(n) en w tal que w = { f(n+1) , f(n) } siendo w el posterior número de Fibonacci para este caso, queremos encontrar la matriz A tal que A * v = w , para cumplir con la regla de Fibonacci y poder extraer de la matriz resultante en la posición cero cero f(n). Así pues, al despejar A vemos que A = { [1, 1] , [1, 0] }

Véase unos ejemplos prácticos:

v = { f(2) , f(1) } => A * v = { [1, 1] , [1, 0] } * { 2, 1 } = {3, 2} siendo 3 el resultado de fib(2)

v = { f(3) , f(2) } => A * v = { [1, 1] , [1, 0] } * { 3, 2 } = {5, 3} siendo 5 el resultado de fib(3)

v = { f(1) , f(0) } => A * v = { [1, 1] , [1, 0] } * { 1, 0 } = {1, 0} siendo 1 el resultado de fib(1)

por lo tanto, se llega a la conclusión que para calcular Fibonacci de un número n, podemos utilizar esta fórmula: A ^ (n-1) * v = { f(n) , f(n-1) }.

Y es con este proceso que se puede calcular Fibonacci de manera concurrente y paralelizar el proceso, haciendo que cada núcleo disponible haga una de estas multiplicaciones matriciales (pudiendo dividir hasta 8 el número de cálculos distintos, véase en el código).

Github

https://github.com/somozadev/Programacion-concurrente/tree/master/ACTIVIDAD_AI4