Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

Reporte Evidencia 2

Ellioth Romero A01781724

Luis Filorio A01028418

Se escribieron tres programas que hacen la misma tarea, pero de manera distinta. Todos los programas revisan todos los archivos dentro de un folder (incluyendo anidaciones), y coleccionan una lista de direcciones donde se hayan encontrado archivos con terminación ".py", pues nuestro lexer funciona con palabras de python. Aquí es donde llegamos a las diferencias, en el programa donde no incluimos procesamiento paralelo, simplemente iteramos sobre la función del lexer por el número de direcciones recolectadas. En el script donde se usó la librería de "multiprocessing", utilizamos un ciclo que por cada directorio encontrado, creará un proceso que se dedicará a analizar n archivo en la lista. Después de crear todos los procesos, un ciclo empezará todos los procesos, y otro ciclo esperará a que terminen todos. Hicimos una prueba extra con un script que usa la librería de "threading", que hace lo mismo que el script anterior, pero con hilos, no núcleos, más adelante explicaremos por qué hicimos esto.

Se midieron los tiempos de múltiples ejecuciones en cada programa:

La solución sin procesamiento paralelo (LexerMulti.py), nos arrojó un tiempo de ejecución promedio de 0.03s.

La solución de procesamiento paralelo utilizando núcleos, nos arrojó un tiempo de ejecución promedio de 0.09s, incluso a veces llegando a 0.11s.

La solución de procesamiento paralelo utilizando hilos nos arrojó un tiempo de ejecución promedio de 0.12s

Los tres resultados dan pie a un análisis interesante. El primer programa es el mejor para usos prácticos, no muy enfocados en la eficiencia ni el tiempo de ejecución. El segundo es el mejor para tareas más pesadas, en nuestro ejemplo solamente utilizamos 5 archivos para analizar, por lo que es algo exagerado usar este método de procesamiento, el overhead de este proceso causa tiempos de ejecución mayores a los demás, pues se tiene que distribuir la tarea entre múltiples núcleos, asignar memoria nueva, proveer de contexto a cada núcleo, lo cual supone una desventaja en el caso de tareas pequeñas, pero para tareas más grandes, es este método el que sale triunfal, el overhead se ve compensado por la eficiencia que pueden proveer los núcleos.

El tercer programa utiliza hilos, es un proceso similar al de los núcleos, pero donde estamos creando tareas distribuidas dentro de un mismo espacio en memoria, lo cual nos provee una alternativa menos exigente en cuanto a recursos, pero menos eficiente para tareas más pesadas. La comunicación rápida que nos proporcionan los hilos es lo que hace que sean tan buena opción para tareas pequeñas.

Todas nuestras funciones tienen una complejidad general de O(n). La búsqueda de archivos .py depende directamente del número de folders y otros archivos dentro del folder. Nuestro ciclos abren un proceso donde se analizan todos los caracteres por cada archivo .py en una lista, seguimos teniendo una complejidad de O(n), donde n es el número total de caracteres.

A simple vista podría parecer que esto no tiene ningún tipo de implicación ética, pero en la realidad, es importante saber utilizar este tipo de herramientas para tener un mejor manejo de recursos en sistemas computacionales, y por lo tanto, un mejor manejo de energía, pieza clave en los principios de sustentabilidad a los que debemos apuntar, como sociedad, a seguir al pie de la letra.