Programación paralela

práctica evaluable
búsqueda en paralelo

Ó. Martín
Facultad de Ciencias Matemáticas
Universidad Complutense de Madrid

febrero de 2024

Tenemos una gran cantidad de números enteros escritos en unos cuantos archivos de texto. Entre todos esos números, queremos encontrar uno (cualquiera) que sea primo y acabe en 227. Prevemos que hay pocos números con esa característica y que nos puede llevar algún tiempo encontrar uno, así que queremos paralelizar el trabajo entre varios procesos. Como solo necesitamos un resultado (un número), en cuanto uno de los procesos encuentre uno válido, queremos que *todos* los procesos dejen de buscar.

Todo esto debe ser programado en Python usando el módulo `multiprocessing`. El proceso principal ha de colocar en una cola (un objeto de la clase `multiprocessing.Queue`) los nombres de los archivos. Para que puedas hacer pruebas, los archivos deben existir en tu ordenador. Se han de crear una cierta cantidad de procesos *buscadores* (elige la cantidad que quieras). Cada proceso buscador ha de tomar de la cola, mientras no esté vacía, un nombre de archivo y procesar su contenido en busca de un elemento que cumpla la condición. Acabado de procesar un archivo, si aún nadie ha encontrado un elemento válido y la cola aún no está vacía, tomará y procesará otro nombre de archivo de la cola. Sugiero el uso de un evento (un objeto de la clase `multiprocessing.Event`) para coordinar los procesos y conseguir que todos dejen de buscar cuando deben. Como resultado visible de la búsqueda, se debe mostrar por pantalla el número que se ha encontrado y el nombre del archivo en el que está.

La condición de ser primo acabado en 227 es solo un ejemplo. La condición concreta en cada caso debe estar escrita en tu programa como una función booleana y pasarse como un argumento a los procesos buscadores. Para más claridad: cada proceso buscador debe recibir, al menos, tres parámetros: una cola de nombres de archivos, una función booleana y un evento.

El recién descrito es el escenario básico. La variante, digamos, *avanzada* consiste en encontrar varios elementos que cumplan la condición, una cantidad dada de ellos. Sugiero que uses en tu programa unas líneas parecidas a estas:

```
nr_buscadores = 4
nr_datos_requeridos = 7
```

Esto permite fijar el valor de los parámetros y facilita cambiarlos entre distintas ejecuciones.

Un equipo de dos personas puede obtener 5 puntos (sobre 10) con una solución excelente al escenario básico. Una solución excelente a la variante avanzada otorga 10 puntos a cada miembro de un equipo de tres personas.

Si en todo este enunciado hay puntos que no están suficientemente precisados, elige lo que te parezca más razonable, más natural, más interesante o más meritorio de resolver; y siempre puedes consultar a tu profesor. Puedes usar en tu solución cualquier herramienta en el módulo `multiprocessing`, incluso si no se han contado en clase. Puedes añadir las explicaciones textuales que creas que pueden ayudar a tu profesor a entender el funcionamiento de tu programa y a apreciar su mérito. Si las explicaciones son de más de tres o cuatro líneas, puede ser mejor ponerlas en un archivo de texto aparte.

Y no se os olvide decirme claramente, por el medio que sea, los nombres de las personas que forman el equipo.