Multiprocessing

José de Jesús Tapia López

27 de Agosto de 2019

1 ¿Qué es y para qué sirve el Multiprocessing?

El multiprocessing es un paquete que admite procesos de generación utilizando una API similar al módulo de threading. El paquete de multiprocesamiento ofrece concurrencia tanto local como remota, evitando de manera efectiva el bloqueo global del intérprete mediante el uso de subprocesos en lugar de threads. Debido a esto, el módulo de multiprocessing permite al programador aprovechar al máximo múltiples procesadores en una máquina determinada. En otras palabras, es la capacidad de un sistema para admitir más de un procesador al mismo tiempo. Las aplicaciones en un sistema de multiprocesamiento se dividen en rutinas más pequeñas que se ejecutan de forma independiente. El sistema operativo asigna estos hilos a los procesadores mejorando el rendimiento del sistema.

2 ¿Por qué usar el Multiprocessing?

Podemos mencionar algunas ventajas:

-Espacio de memoria separado.
-El código es generalmente sencillo.
-Aprovecha múltiples CPUs y núcleos.
-Evita las limitaciones de GIL para cPython.
-Elimina la mayoría de las necesidades de primitivas de sincronización a menos que use una memoria compartida (en su lugar, es más bien un modelo de comunicación para IPC).
-Los procesos hijo son interrumpibles / killables.
-El módulo de multiprocessing Python incluye abstracciones útiles con una interfaz muy parecida a la de threading.Thread.
-Contiene equivalentes de todas las primitivas de sincronización de threading.

3 ¿Cómo sirve y cómo utilizo el multiprocesamiento en Python?

En el multiprocesamiento, los procesos se generan creando un objeto Process y luego llamando a su método start() . El proceso sigue la API de subprocesamiento.

Dependiendo de la plataforma, el multiprocesamiento admite tres formas de iniciar un proceso. Estos métodos de inicio son:

Spawn: El proceso padre inicia un nuevo proceso de intérprete de Python. El proceso hijo solo heredará los recursos necesarios para ejecutar el método run() de los objetos de proceso. En particular, no se heredarán los descriptores de archivo innecesarios y los identificadores del proceso principal. Iniciar un proceso usando este método es bastante lento en comparación con el uso de fork o forkserver.

Fork: El proceso padre usa os.fork () para bifurcar el intérprete de Python. El proceso hijo, cuando comienza, es efectivamente idéntico al proceso padre. Todos los recursos del padre son heredados por el proceso hijo. Se debe tomar en cuenta que bifurcar de forma segura un proceso multiproceso es problemático.

Forkserver: Cuando el programa se inicia y selecciona el método de inicio de forkerserver se inicia un proceso de servidor. A partir de ese momento, cada vez que se necesite un nuevo proceso, el proceso padre se conecta al servidor y solicita que bifurque un nuevo proceso. El proceso del servidor fork es de un solo threaded, por lo que es seguro usar os.fork () . No se heredan recursos innecesarios.

Nota: Los métodos de inicio 'spawn' y 'forkserver' no se pueden usar actualmente con ejecutables "congelados".

El paquete Multiprocessing admite dos tipos de canales de comunicación entre procesos:
-La clase Queue, que es un clon cercano de queue.Queue.
-La función Pipe(), que devuelve un par de objetos de conexión conectados por una tubería que por defecto es dúplex (bidireccional).

Cuando se realiza una programación concurrente, generalmente es mejor evitar el uso del estado compartido en la medida de lo posible. Esto es particularmente cierto cuando se utilizan múltiples procesos. Sin embargo, si realmente necesita utilizar algunos datos compartidos, el multiprocessing proporciona un par de formas de hacerlo:

-Memoria compartida: Los datos se pueden almacenar en un mapa de memoria compartida usando Value o Array.
-Proceso del servidor: Un objeto de administrador devuelto por Manager() controla un proceso de servidor que contiene objetos de Python y permite que otros procesos los manipulen usando proxies. Un administrador devuelto por Manager() admitirá tipos list, dict, Namespace, Lock, RLock, Semaphore, BoundedSemaphore Semaphore, Condition, Event, Barrier, Queue, Value y Array.