CC-Tarea1

Antonio Hernández

Octubre 2020

2: Librerías y funciones para el cómputo concurrente en C

1. `<pthread.h>`

   Librería para manejar hilos POSIX

2. `pthread_create()`

   La función se usa para crear un nuevo hilo, con atributos especificados por attr, dentro de un proceso. Si attr es NULL, se utilizan los atributos predeterminados. Una vez completado con éxito, pthread_create () almacena el ID del hilo creado en la ubicación a la que hace referencia el hilo.

3. `pthread_join()`

   Espera a que el hilo que recibe como parámetro termine.

4. `pthread_t`

   Se otorga un nombre de identificación al hilo.

5. `pthread_exit()`

   La función termina el hilo que tome por parámetro.

6. `fork()`

   Crea un proceso y devuelve su id

7. `getpid()`

   Devuelve el identificador de proceso del proceso actual

8. `getppid()`

   Devuelve el identificador de proceso del padre del proceso actual.

9. `<unistd.h>`

   Biblioteca estándar de llamadas al sistema

10. `pipe()`

    Permite hacer comunicación entre dos procesos.

11. `close()`

    Cierra el canal de lectura o escritura según sean los parámetros que reciba.

12. `getuid()`

    Regresa el ID del propietario que inicio el proceso

13. `<sys/wait.h>`

    Biblioteca para hacer llamada a wait

14. `wait()`

    Toma por parámetro un puntero a un entero que contendrá el estado de salida de ese programa. Devuelve el ID de proceso del hijo que terminó.

15. `pid_t`

    Retorna el identificador del proceso

3: Algunas definiciones de cómputo concurrente.

1. *Global Interpreter Lock (GIL)*:
   Es un bloqueo que permite que solo un hilo mantenga el control del intérprete de Python. Esto significa que solo un subproceso puede estar en estado de ejecución en cualquier momento.

   El GIL no tiene mucho impacto en el rendimiento de los programas de subprocesos múltiples enlazados a E/S ya que el bloqueo se comparte entre los subprocesos mientras esperan la misma.

   Pero un programa cuyos subprocesos están completamente vinculados a la CPU, por ejemplo, un programa que procesa una imagen en partes usando subprocesos, no solo se volverá un solo subproceso debido al bloqueo, sino que también verá un aumento en el tiempo de ejecución.

2. *Ley de Amdhal*:
   Evalúa como cambia el rendimiento al mejorar una parte de la computadora.
   La fórmula original de la ley de Amdahl es la siguiente

$$T_m = T_a \cdot \left((1 - F_m) + \frac{F_m}{A_m} \right) \tag{1}$$

   Siendo:
   F_m =fracción de tiempo que el sistema utiliza el subsistema mejorado.
   A_m =factor de mejora que se ha introducido en el subsistema mejorado.
   T_a =tiempo de ejecución antiguo.
   T_m =tiempo de ejecución mejorado.

3. *multiprocessing*: El paquete de multiprocesamiento ofrece simultaneidad local y remota, evitando efectivamente el bloqueo de intérprete global mediante el uso de subprocesos en lugar de hilos. Debido a esto, el módulo de multiprocesamiento permite al programador aprovechar al máximo múltiples procesadores en una máquina determinada. Funciona tanto en Unix como en Windows.

   a) run(): Método que representa la actividad del proceso.
   Puede anular este método en una subclase. El método estándar run () invoca el objeto invocable pasado al constructor del objeto como el argumento de destino, si lo hay, con argumentos secuenciales y de palabras clave tomados de los argumentos args y kwargs, respectivamente.

   b) start(): Inicie la actividad del proceso.
   Esto se debe llamar como máximo una vez por objeto de proceso. Organiza que el método run () del objeto sea invocado en un proceso separado.

   c) join([timeout]): Bloquea el hilo de llamada hasta que finalice el proceso cuyo método join () se llama o hasta que se agote el tiempo de espera opcional.
   Si el tiempo de espera es Ninguno, no hay tiempo de espera.
   Un proceso se puede unir muchas veces.
   Un proceso no puede unirse a sí mismo porque esto provocaría un punto muerto. Es un error intentar unirse a un proceso antes de que se haya iniciado.

   d) pid: Devuelve el ID del proceso. Antes de que se genere el proceso, esto será nulo.

   e) terminate(): Termina el proceso

4: Crear procesos como un objeto que hereda de la clase multiprocessing.process

```
def f(name):
info('function f')
print('hello', name)

if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

Referencias

https://realpython.com/python-gil/
https://www.geeksforgeeks.org/pipe-system-call/
http://manpages.ubuntu.com/manpages/bionic/es/man2/getpid.2.html
https://pubs.opengroup.org/
http://profesores.elo.utfsm.cl/ agv/elo330/2s07/lectures/ControlProcesos.html
https://docs.python.org/2/library/multiprocessing.html