Semafori in Python

Confronto tra System V e POSIX

Alessio Rubicini - 4INC - I.T.T. Montani Fermo
2019/2020

Codici di esempio:
github.com/alessiorubicini/Unix-IPC-in-Python

Moduli

Sistemi di IPC System V: https://pypi.org/project/sysv_ipc/

Sistemi di IPC POSIX: https://pypi.org/project/posix_ipc/

Per installarli:

```
python3 -m pip install sysv_ipc

python3 -m pip install posix_ipc
```

Creazione del semaforo (SYSTEM V)

```
sem = Semaphore(key, flag, mode, initial_value)
```

Crea o apre un semaforo (in base al parametro flag) identificato dalla chiave key e con valore initial_value

Parametri:

- **Key**: Chiave del semaforo, deve essere None, IPC_PRIVATE o un intero compreso tra -9223372036854775807 e 9223372036854775807.
  Se è None il modulo sceglie una chiave random non utilizzata
- **Flag**: 0 (default) apre un semaforo esistente, IPC_CREAT apre o crea il semaforo se non esiste, IPC_CREX crea un semaforo e restituisce errore *(ExistentialError)* se già esiste
- **Mode**: permessi (ignorati quando si apre un semaforo già esistente)
- **Initial_value**: valore a cui viene inizializzato il semaforo

Creazione del semaforo (POSIX)

```
sem = Semaphore(name, flag, mode, initial_value)
```

Crea o apre un semaforo (in base al parametro flag) identificato dalla chiave key e con valore initial_value

Parametri:

- **Name**: nome del semaforo. Deve essere *None* o una stringa. Se è *None* il modulo sceglie automaticamente un nome non utilizzato, se è una stringa deve iniziare con uno slash ed essere valido in base alle regole dei path del proprio sistema
- **Flag**: 0 (default) apre un semaforo esistente, IPC_CREAT apre o crea il semaforo se non esiste, IPC_CREX crea un semaforo e restituisce errore *(ExistentialError)* se già esiste
- **Mode**: permessi (ignorati quando si apre un semaforo già esistente)
- **Initial_value**: valore a cui viene inizializzato il semaforo

Acquisizione semaforo (SYSTEM V)

```
sem.acquire(timeout, delta)
```

Aspetta che il valore del semaforo sia maggiore di 0 e ritorna decrementandolo

Parametri:

- timeout: specifica quanto deve aspettare. Se è None non c’è nessun limite di tempo, se è 0 ritorna subito errore nel caso non lo possa acquisire, se è maggiore di 0 aspetta i secondi specificati
- delta: quantità di incremento del semaforo

Acquisizione semaforo (POSIX)

```
sem.acquire(timeout)
```

Aspetta che il valore del semaforo sia maggiore di 0 e ritorna decrementandolo

Parametri:

- timeout: specifica quanto deve aspettare. Se è None non c’è nessun limite di tempo, se è 0 ritorna subito errore nel caso non lo possa acquisire, se è maggiore di 0 aspetta i secondi specificati

Rilascio semaforo (SYSTEM V)

```
sem.release(delta)
```

Rilascia il semaforo incrementando il suo valore

Parametri:

- Delta: specifica di quanto decrementare il semaforo, di default è 1

Rilascio semaforo (POSIX)

```
sem.release()
```

Rilascia il semaforo incrementando il suo valore

Nessun parametro

Nomenclature alternative per i metodi

Nei semafori System V esistono dei metodi “sinonimi” di quelli menzionati precedentemente che hanno nomi abbreviati rispetto a quelli originali ma il medesimo funzionamento.

ORIGINALE	ABBREVIAZIONE
.acquire()	P() Si chiama P come abbreviazione di “*prolaag or probeer te verlagen”* dall’olandese “prova a diminuire” (nome originale dato da Edsger Dijkstra)
.release()	V() Si chiama V *come abbreviazione di “verhoog”* dall’olandese “aumentare” (nome originale dato da Edsger Dijkstra)

Rimozione semaforo (SYSTEM V)

```
sem.remove()
```

Rimuove il semaforo dal sistema.

Il comportamento del metodo quando il semaforo viene eliminato mentre altri processi lo stanno utilizzando dipende dal sistema operativo.

Si consiglia perciò di controllare le pagine del manuale nella sezione relativa alla *semctl(IPC_RMID).*

Chiusura e Rimozione semaforo (POSIX)

`sem.close()`

Chiude il semaforo indicando che il processo corrente ha terminato di lavorare con esso.

`sem.unlink()`

Distrugge il semaforo. Se il semaforo è ancora utilizzato da altri processi *unlink()* ritorna e la distruzione viene rimandata fino a quando tutti i processi non hanno chiuso il semaforo.

Documentazione completa

http://semanchuk.com/philip/sysv_ipc/#semaphore

http://semanchuk.com/philip/posix_ipc/#semaphore

Il modulo asyncio

Modulo compreso nella libreria standard di Python che fornisce l'infrastruttura per la scrittura di codice concorrente a thread singolo utilizzando coroutine, eventi, accesso I/O multiplexing su socket e altre risorse, esecuzione di client e server di rete e altre primitive correlate.

https://docs.python.org/3.6/library/asyncio.html

Semafori con asyncio

SEMAFORI (SEMAPHORES)	SEMAFORI “LIMITATI” (BOUNDED SEMAPHORES)
Classici semafori come quelli visti precedentemente sem = Semaphore() sem.acquire() sem.release() sem.locked()	Questi particolari semafori restituiscono errore se il proprio valore viene incrementato oltre il valore a cui sono stati inizializzati (initial value).

E per i thread?

Ci sono 2 opzioni:

1) Utilizzare i semafori come mostrato precedentemente
2) Utilizzare gli oggetti lock del modulo `threading`

Sincronizzazione mediante threading.Lock()

Gli oggetti *Lock* sono la primitiva di sincronizzazione più semplice e più a basso livello nel linguaggio Python. Sono compresi nel modulo threading della libreria standard e il loro utilizzo in termini di metodi è quasi identico a quello dei semafori.

```
# Creazione oggetto lock
mutex = threading.Lock()

# Acquisizione lock
mutex.acquire()

# Rilascio lock
mutex.release()
```