GUI multithreaded

Produttori e consumatori

- In generale, una GUI è implementata attraverso una applicazione multithreaded
  - Un thread produce dati
  - Un thread riceve i dati e aggiorna la GUI
- Modello di tipo produttore/consumatore
- Tale modello è implementato attraverso l'uso di una coda di comunicazione

Il modulo Queue

- Il modulo Queue implementa un semplice meccanismo di comunicazione a messaggi basato su code

```
import Queue
```

- Una coda è definita attraverso l'istanza di un oggetto di tipo Queue

```
q = Queue.Queue()
```

- L'oggetto di tipo Queue è già serializzato internamente
- È possibile impostare una dimensione massima della coda mediante il parametro maxsize

```
q = Queue.Queue(maxsize=10)
```

Il modulo Queue

- Metodi utili
  - get(): estrae un elemento dalla coda
  - put(): inserisce un elemento nella coda
  - qsize(): ritorna il numero di elementi in coda
  - empty(): ritorna True se la coda è vuota, False altrimenti
  - full(): ritorna True se la coda è piena, False altrimenti

Un primo, semplice esempio

- Inseriamo cinque elementi in una coda
- Preleviamo elementi fino a quando la coda non si svuota
- Si esegua l'esempio queue.py
- Scopriremo che la disciplina di inserimento è la FIFO
- Esistono altre discipline
  - LIFO (stack)
  - Priority

Code LIFO (stack)

- ESEMPI: queue_LIFO.py

- È possibile definire una disciplina di tipo LIFO istanziando un oggetto di tipo LifoQueue

```
q = Queue.LifoQueue()
```

- Si esegua l'esempio queue_LIFO.py e si osservi la differenza con l'esempio precedente

Code di priorità

- ESEMPI: queue_PRIO.py

- È possibile definire una disciplina di tipo Priority istanziando un oggetto di tipo PriorityQueue

```
q = Queue.PriorityQueue()
```

- Una coda con priorità riceve ed estrae oggetti aventi un metodo __cmp__()
- Il metodo __cmp__() viene usato
  - per ordinare gli elementi in senso crescente
  - per restituire l'elemento con la priorità più bassa
- Gli oggetti dovrebbero anche avere informazioni legate al livello di priorità
  - Attributo della classe

Uso delle code con i thread

- Le code possono essere condivise fra thread produttori e consumatori
  - Implicitamente, usando una variabile globale di tipo coda
  - Passando l'oggetto coda nei costruttori degli oggetti Thread
- Quest'ultima soluzione è più pulita (produttori e consumatori possono essere in file diversi)

Uso delle code con i thread

- join()
  - Si blocca fino a quando tutti gli elementi della coda non sono stati estratti ed elaborati
- task_done()
  - Usato dai thread consumatori
  - In seguito ad una get(), task_done() comunica alla coda che la procedura di consumo è terminata
  - Quando ciò avviene per tutti gli elementi della coda, join() esce
- Meccanismo analogo ai due semafori suppletivi (vuote, piene) del classico problema produttore/consumatore

Uno scheletro di coordinazione

```
def worker():
    while True:
        item = q.get()
        do_work(item)
        q.task_done()

q = Queue()
for i in range(num_worker_threads):
    t = Thread(target=worker)
        t.daemon = True
        t.start()

for item in source():
    q.put(item)

q.join()
```

Un esempio di coordinazione

- Scriviamo un fetcher multithreaded di immagini da un URL (imagefetch.py)
- Usiamo un numero di produttori configurabile da linea di comando (--num-threads)
- Usiamo il modulo HTMLParser per estrarre gli URL delle immagini
- Usiamo una coda FIFO per memorizzare gli URL
- Usiamo il modulo urllib2 per scaricare le immagini
- task_done() e join() assicurano l'avanzamento ed il termine dell'applicazione

Thread e GTK+

- ESEMPI: imagefetch.py

- La libreria GTK+ è thread-aware, ma non è thread-safe
- Thread-awareness
  - Viene fornito un lock globale, in modo tale che un thread alla volta possa usufruire delle funzioni di GTK+
  - Acquisizione: gtk.gdk.threads_enter()
  - Rilascio: gtk.gdk.threads_leave()
- Thread-safety
  - Nessuna; i lock vanno messi esplicitamente
  - Il meccanismo di lock va inizializzato mediante la chiamata gtk.gdk.threads_init()

Scheletro di una GUI multi threaded

- Si inizializza il locking: gtk.gdk.threads_init()
- Si crea una classe producer, la quale implementa in un thread una operazione di produzione contenuti
- Si crea una classe consumer, la quale implementa in un thread una operazione di consumo dei contenuti
- Si usa una coda per far comunicare le due classi
- Si aggiorna la GUI al momento della ricezione
  - L'aggiornamento avviene in mutua esclusione, tramite il GTK+ global lock

Timeout e idle call

- La classe consumatrice può essere sostituita con una funzione invocabile
  - periodicamente (timeout)
  - quando il main loop GTK+ non ha nulla da fare (idle call)
- Lo scopo di tale funzione è quello di aggiornare la GUI con i nuovi contenuti
- Se i contenuti prodotti sono diversi, è preferibile usare diverse classi consumatrici, le quali invocano le funzioni di aggiornamento della GUI in mutua esclusione

Timeout call

- Le timeout call sono fornite dal modulo gobject

```
import gobject
```

- L'aggiunta di un timer avviene tramite il metodo gobject.timeout_add(), che ritorna un identificatore del timer appena imposto

```
timer_id = gobject.timeout_add( \
    timeout_msec, \ method)
```

- La rimozione di un timer avviene tramite il metodo gobject.source_remove()

```
gobject.source_remove(timer_id);
```

Idle call

- Le idle call sono fornite dal modulo gobject

```
import gobject
```

- L'aggiunta di una idle call avviene tramite il metodo gobject.idle_add()

```
gobject.idle_add(method, args)
```

- Se il metodo ritorna True, sarà invocato di nuovo in futuro
- Se il metodo ritorna False, viene cancellato dalla lista degli eventi e non sarà più invocato

Un esempio di GUI

- ESEMPI: ThreadGUI.py

- Scriviamo una GUI multi threaded che aggiorna una text view con un testo costante
- La text view è provvista di scrollbar orizzontali e verticali, per contenere un testo più largo dell'ampiezza della finestra
- Thread Producer
  - Produce il messaggio
  - Inserisce il messaggio nella coda
- ThreadGUI
  - Inizializza l'interfaccia grafica
  - Estrae periodicamente un messaggio dalla coda e lo stampa nella text view