Gestione delle risorse

Una problematica rilevante quando si sviluppa software è quella di riuscire ad ottenere una corretta gestione delle risorse.

> Con il termine "risorse" indichiamo genericamente entità che, intuitivamente, sono disponibili in quantità limitata e che, in caso di esaurimento, potrebbero compromettere o limitare la funzionalità del software. Per questo motivo, il software deve necessariamente interagire con le risorse in modo corretto, evitando che alcune di esse vadano "perse" o siano compromesse rendendole inutilizzabili.

In linea di massima, l'interazione del software con le risorse deve avvenire secondo uno schema predefinito, suddiviso in tre fasi ordinate temporalmente:

1. acquisizione della risorsa
2. uso della risorsa
3. restituzione (rilascio) della risorsa, perché le risorse disponibili sono limitate

> È molto importante non utilizzare una risorsa prima di averla acquisita e dopo averla rilasciata.

In particolare:

- la risorsa deve essere acquisita prima di essere usata (o rilasciata);
- al termine del suo utilizzo la risorsa deve essere rilasciata;
- non è lecito usare una risorsa dopo averla rilasciata.

Torna all'indice

Esempi di risorse

La memoria ad allocazione dinamica

Viene acquisita tramite l'uso dell'espressione new (magari effettuato indirettamente), utilizzata per leggere e scrivere valori (mediante dereferenziazione di un puntatore) e infine rilasciata mediante l'uso della delete (magari effettuato indirettamente).

Il mancato rilascio genera memory leak e, in casi particolari, può portare all'esaurimento della memoria disponibile. L'accesso a dangling pointer è un esempio di uso dopo il rilascio. La double free è un esempio di rilascio di una risorsa che non è più sotto il nostro controllo (già rilasciata in precedenza).

> Il termine "memory leak", che vuol dire "perdita o fuoriuscita di memoria", indica un particolare consumo non voluto di memoria causato dalla mancata deallocazione di variabili/dati non più necessari.

Torna all'indice

I (descrittori dei) file del file system

Vengono acquisiti con l'operazione di apertura del file, utilizzati per leggere e/o scrivere sul file e infine rilasciati con l'operazione di chiusura. In casi specifici, la mancata operazione di chiusura di un file potrebbe comprometterne il contenuto.

Torna all'indice

I lock per l'accesso (condiviso o esclusivo) a risorse condivise

L'acquisizione dei lock è necessaria per una corretta gestione della condivisione delle risorse (da parte di più processi o thread); il mancato rilascio di un lock può causare deadlock o starvation.

Esempio: le connessioni di rete a server (e.g., sessioni DBMS).

> Per semplicità di esposizione, useremo spesso il caso della memoria dinamica (perché è quello più semplice da simulare negli esempi), ma deve essere chiaro che il discorso vale in generale.

Torna all'indice

Exception safety

Una gestione corretta delle risorse è tutto sommato semplice da ottenere quando "tutto fila liscio", cioè quando il nostro software segue uno dei flussi di esecuzione previsti dal programmatore.

La situazione tende però a complicarsi non appena si ammette la possibilità che qualcosa possa "andare storto", ovvero quando alcune delle operazioni eseguite possono incorrere in situazioni di errore che, pur essendo state previste in linea di principio, non sono state (o non possono essere) gestite esplicitamente.

Nel caso del C++, la tecnica idiomatica per segnalare situazioni di errore consiste nel lanciare eccezioni, facendo quindi in modo che il programma esca dal normale flusso di esecuzione e entri nei cosiddetti flussi di esecuzione "eccezionali".

Quando eseguite in modalità eccezionale, quasi tutte le strutture di controllo del programma (concatenazione, costrutti condizionali, construtti iterativi, ecc.) si comportano in maniera diversa dal normale, tipicamente ignorando larga parte del codice scritto dal programmatore.

Una volta lanciata un'eccezione, questa si propaga lungo la catena delle chiamate e l'unico modo di rientrare nel flusso di esecuzione normale è di catturare l'eccezione (in un blocco `catch`) e gestirla; in assenza di un blocco `catch` adeguato, il programma giungerà a terminazione in modalità eccezionale.

Diventa quindi essenziale capire nel dettaglio cosa succede in questi casi; in particolare, occorre acquisire le tecniche di programmazione che consentono di ottenere una corretta gestione delle risorse anche in presenza di comportamenti eccezionali del codice.

Questo è l'obiettivo della cosiddetta *exception safety*.

Un esempio banale:

```
void foo() {
 int* pi = new int(42);
 do_the_job(pi);
 delete pi;
}
```

La funzione `foo` acquisisce una risorsa (la memoria dinamica puntata dal puntatore `pi`), la passa alla funzione `do_the_job` (che la usa) e infine la rilascia mediante la `delete`. In condizioni normali, `foo` gestisce correttamente la risorsa in questione, perché le tre fasi (acquisizione, uso, rilascio) si susseguono secondo l'ordine corretto.

La funzione `foo`, però, non è exception safe: se la funzione `do_the_job` (o una qualunque delle funzioni invocate da questa) lancia una eccezione e tale eccezione non viene gestita internamente, il flusso di esecuzione in uscita dalla chiamata è un flusso eccezionale, che NON esegue l'istruzione `delete pi`. Quindi, si uscirebbe dall'esecuzione di `foo` (in modalità eccezionale) senza avere rilasciato la risorsa, ottenendo un *memory leak*.

Dal punto di vista metodologico, la domanda che ci dobbiamo fare è quindi la seguente: quali sono le tecniche che possiamo utilizzare per modificare il codice della funzione `foo` affinché diventi exception safe? Tra le varie tecniche utilizzabili, ve ne sono alcune migliori di altre che, pertanto, sarebbe raccomandato seguire?

> NOTA BENE: esistono contesti nei quali è perfettamente accettabile scrivere codice che non sia exception safe, ovvero codice che NON gestisce correttamente le risorse in presenza di comportamenti eccezionali.
>
> Per esempio, questo capita quando:

1. La risorsa in questione è poco importante.
2. La correttezza del software (nei casi in cui si presenta un comportamento eccezionale) è di scarso interesse.
3. Il tempo di esecuzione del software è molto breve.

L'exception safety assume rilevanza quando almeno una delle condizioni suddette non è vera.

Torna all'indice