Jupiter Festival Lined up Splashy Sponsors but Had a Soft Landing In Its First Year

Organizers say 2,231 attendees met their goal but the show felt sparse at times. https://www.adweek.com/media/jupiter-festival-lined-up-splashy-sponsors-but-had-a-soft-landing-in-its-first-year/

The FileCopy – come scrivere una funzione di File Copy in C (e Go) – pt.3

R.J.MacReady: *Qualcuno in questo campo non è quello che sembra. Al momento potrebbe trattarsi di uno o due di noi. Entro la primavera, potremmo essere tutti noi.*
Childs: *Allora, come facciamo a sapere chi è umano? Se fossi un'imitazione, un'imitazione perfetta, come faresti a capire se fossi davvero io?*

Dove eravamo rimasti? Ah, si! **Nell'ultimo articolo** (*l'avete letto vero?*) avevo proposto la versione *buffered* della funzione **cpFile()**, dopo aver proposto (**nel penultimo articolo**) la versione *unbuffered* e quella *"special"* che funziona in *kernel-space*. E oggi, come promesso (*e le promesse si mantengono, sempre*) vi racconterò di come si può fare in **Go** la stessa cosa. Ma è doveroso, anche per questa terza puntata,

fornire un altra immagine tratta da **The Thing** del Maestro **John Carpenter**, per ricordare che questo, come ben sapete, è un blog per programmatori cinefili…



…le copie dei file vengono meglio col fresco…

Ok, veniamo al dunque: ho scritto due versioni in Go della **cpFile()**, la prima si può accomunare ai tipi *buffered/unbuffered* già visti, mentre la seconda si può paragonare al tipo *"special"*. Cominciamo con la versione *"normal"*, che ho chiamato *cpbase.go*. Vai col codice!

```
// cpbase.go - un cp(1) con os.ReadFile() e os.WriteFile()
package main import ( "os" "fmt"
) // funzione main
func main() { // test argomenti argc := len(os.Args) if (argc
!= 3) { // errore: numero errato di argomenti: esco
fmt.Printf("%s: numero errato di argomenti\n", os.Args[0])
fmt.Printf("uso: %s src dest [e.g.: %s prova.c prova.bak]\n",
os.Args[0], os.Args[0]) os.Exit(1) } // eseguo la copia
n_copied, err := cpFile(os.Args[2], os.Args[1]) if err != nil
{ // errore: copia fallita: esco fmt.Printf("%s: error: %s\n",
os.Args[0], err); os.Exit(1) } // esco con Ok fmt.Printf("%s:
copiati %d byte\n", os.Args[0], n_copied); os.Exit(0)
} // cpFile() - copia il file <src> nel file <dest>
func cpFile(dest string, src string) (n_copied int, err error)
{ // leggo tutto il contenuto dal file sorgente e lo metto in
<data []byte> // NOTA: non usare per file molto grandi:
potrebbe non bastare la memoria data, err := os.ReadFile(src)
if err != nil { // errore: ritorno 0 e l'errore return } //
```

```
scrivo il contenuto sul file destinazione (specificando i permessi) err = os.WriteFile(dest, data, 0644) if err != nil { // errore: ritorno 0 e l'errore return } // set byte copiati n_copied = len(data) // ritorno il numero di byte copiati ed errore nullo return
}
```

Come avrete sicuramente notato (e letto nei **dettagliatissimi commenti**) il *main()* è praticamente identico a quello dei programmi **C** visti precedentemente, mentre la funzione *cpFile()*, pur comportandosi esattamente nello stesso modo di quella in C, è alquanto diversa e, direi, molto semplificata: leggo il sorgente (prima operazione), scrivo sulla destinazione (seconda operazione) e ricavo il numero di *byte* copiati (terza operazione). Praticamente sono tre linee più i tradizionali test degli errori tipici del Go, tutto qua! Il Go conferma ancora una volta di essere un vero linguaggio di alto livello.

Ma c'è un *"ma".* Avrete sicuramente visto la nota:

// NOTA: non usare per file molto grandi: potrebbe non bastare la memoria

Ecco, questa versione è adatta solo per *file* relativamente piccoli, perché il *file,* prima di essere copiato nella destinazione, viene caricato completamento nello *slice di byte data* (di tipo *[]byte*) che è allocato nello *heap:* e *l'heap* ha una dimensione limitata, ragionevolmente proporzionale alla memoria fisica disponibile, quindi se il file è molto grande (*chessoio, 100GB*) avremo un problema di memoria. Quindi questa funzione è imperfetta: non funziona con qualsiasi file e quando non funziona genera un grave errore di memoria. Se proprio volete usarla, come minimo dovrete aggiungere un controllo preliminare sulla dimensione del *file* sorgente per ritornare, eventualmente, un errore del tipo *"errore: il file è troppo grande".*

E allora passiamo alla versione *"special"* quella che risolve

il problema appena mostrato: vai col codice!

```
// cpspecial.go - un cp(1) con io.Copy()
package main import ( "io" "os" "fmt"
) // funzione main
func main() { // test argomenti argc := len(os.Args) if (argc
!= 3) { // errore: numero errato di argomenti: esco
fmt.Printf("%s: numero errato di argomenti\n", os.Args[0])
fmt.Printf("uso: %s src dest [e.g.: %s prova.c prova.bak]\n",
os.Args[0], os.Args[0]) os.Exit(1) } // eseguo la copia
n_copied, err := cpFile(os.Args[2], os.Args[1]) if err != nil
{ // errore: copia fallita: esco fmt.Printf("%s: error: %s\n",
os.Args[0], err); os.Exit(1) } // esco con Ok fmt.Printf("%s:
copiati %d byte\n", os.Args[0], n_copied); os.Exit(0)
} // cpFile() - copia il file <src> nel file <dest>
func cpFile(dest string, src string) (n_copied int64, err
error) { // apro il file sorgente fsrc, err := os.Open(src) if
err != nil { // errore: ritorno 0 e l'errore return } defer
fsrc.Close() // chiusura differita // apro il file
destinazione fdest, err := os.Create(dest) if err != nil { //
errore: ritorno 0 e l'errore return } defer fdest.Close() //
chiusura differita // copio da sorgente a destinazione usando
io.Copy() // NOTA: è il metodo più efficiente perché usa uno
stream invece della memoria n_copied, err = io.Copy(fdest,
fsrc) if err != nil { // errore: ritorno 0 e l'errore return }
// ritorno il numero di byte copiati ed errore nullo return
}
```

Interessante, no? Il *main()* è, ovviamente, invariato, mentre la funzione *cpFile()* è radicalmente diversa, pur mantenendo la grande compattezza di quella appena vista. Anche in questo caso sono tre operazioni: apro il file sorgente, apro (o creo, se necessario) il file destinazione e copio usando la funzione **io.Copy()** che genera uno *stream* di dati da un file all'altro e si ferma quando raggiunge l'*EOF*: Una *cpFile()* pulita, compatta, super efficiente… e senza limitazioni sulla dimensione dei *file!* Mi piace.

E anche questa terza (e ultima) parte dell'articolo si può considerare conclusa. Il Go si dimostra, ancora una volta, una perfetta controparte ad alto livello del C (e non solo). Ma

ricordate: tutti e due hanno ampie ragioni di essere: il C è, da sempre, il Re de linguaggi a basso livello *multi-purpose* (per *kernel, firmware,* librerie e funzioni di sistema, ecc.), mentre il Go è un fantastico linguaggio ad alto livello più adatto ad altri usi *(microservizi,* API Web, infrastrutture Cloud, ecc.).

E nel prossimo articolo… boh, ci devo ancora pensare. Restate sintonizzati!

Ciao, e al prossimo post!

P.S.

Dopo anni di tentennamenti ho, finalmente, deciso di aprire un *cineblog!* Se vi interessa lo trovate qui: **Polvere di Cinema**. Buona lettura!

https://italiancoders.it/the-filecopy-come-scrivere-una-funzione-di-file-copy-in-c-e-go-pt-3/

Ticker: Former WH Press Secretary Karoline Leavitt Joins Fox News

Leavitt starts her new role in November. https://www.adweek.com/tvnewser/ticker-former-wh-press-secretary-karoline-leavitt-joins-fox-news/

Health Marketing Is Moving From Information to Navigation

This post was created in partnership with Publicis Health Key takeaways A health journey can now begin long before a formal moment of care, sparked by a smartwatch vibration, a […] https://www.adweek.com/brand-marketing/health-marketing-is-moving-from-information-to-navigation/

How MLB Made an Urgent Pitch to Fans for the 2026 Playoffs

The MLB playoffs are seeing a ratings boost, and some of the credit likely goes to the league’s ‘No Time Like October’ campaign. https://www.adweek.com/convergent-tv/how-mlb-made-an-urgent-pitch-to-fans-for-the-2026-playoffs/

Brands Can Expect a Longer Holiday Shopping Season and

Thriftier Consumers

This post was created in partnership with Ibotta Elevated grocery prices and anticipated holiday financial stress are already making shoppers more selective about which brands earn a spot in […] https://www.adweek.com/commerce/brands-can-expect-a-longer-holiday-shopping-season-and-thriftier-consumers/

People Are Getting Together—Will Brands Meet Up With Them?

This post was created in partnership with Atmosphere TV Consumers are forgoing solitary time for social experiences, spending time in public spaces like bars, gyms, and restaurants. This shift presents […] https://www.adweek.com/brand-marketing/people-are-getting-togetherwill-brands-meet-up-with-them/

Building the Right Team Is More Important Than Ever for

Marketers

This post was created in partnership with Weber Shandwick As AI streamlines workflows and marketing budgets tighten, human collaboration and partnership remain as important as ever to create strong creative. […] https://www.adweek.com/brand-marketing/building-the-right-team-is-more-important-than-ever-for-marketers/

Award-Winning Agencies Talk TikTok

This post was created in partnership with TikTok for Business Media buying should be more than placing ads—it should help brands connect with audiences, navigate a changing media landscape, and […] https://www.adweek.com/agencies/award-winning-agencies-talk-tiktok/

Walmart Tests Placing Marketplace Inventory In Stores For Pickup and

Delivery

Items from third-party sellers could soon arrive alongside shoppers' grocery orders. https://www.adweek.com/commerce/walmart-tests-placing-marketplace-inventory-in-stores-for-pickup-and-delivery/