Progetto Sistemi Operativi

Errori comuni

Enrico Bini

December 5, 2018

1 Errori

Segue un elenco di errori comuni che comportano un'errato funzionamento del codice.

1.1 Errata allocazione della memoria

1. È stata eseguita una

```
int * p;

p = malloc(20);
```

   ma poi vengono scritti o letti più byte di quanti allocati (20 nell'esempio).

2. Il caso di cui al punto 1 capita, per esempio, con funzioni che leggono stringhe da uno stream (`stdin` o altro) che non pongono un limite al numero di byte che possono essere letti, come `sprintf(...)`

3. Dimenticare di chiudere una stringa con il byte `'\0'`. Quando una sequenza di byte viene interpretata come "stringa di caratteri", per esempio quando:

   - si stampa con `printf` specificando `%s`
   - si usano funzioni che manipolano stringhe come: `strlen(...)`, `strcat(...)`

1.2 Errata deallocazione della memoria

Eseguire una `free(p)` su un puntatore `p` che non è stato ritornato da una `malloc(...)`/`calloc(...)` genera un errore della `free(p)`. Solitamente, quando questo errore capita quando `p` è stato inavvertitamente modificato altrove. Per esempio con una

```
p++;                /* incrementa il puntatore p */
```

invece che una

```
*p++;     /* incrementa la memoria puntata da p */
```

2 Cattive abitudini

Segue un elenco di "cattive abitudini" ovvero di pratiche che non comportano errori nell'esecuzione, ma che comunque sarebbero da evitare.

2.1 Attesa attiva

Un processo che deve attendere il verificarsi di un evento si dovrebbe mettere in stato "sospeso" attraverso una chiamata opportuna (`wait(...)`, `waitpid(...)`, `read(...)` su una pipe, `msgrev(...)` su una coda, `semop(...)` su un semaforo, `sleep()`, etc.). Se invece, rimane in esecuzione facendo polling in attesa che una condizione diventi vera, per esempio con

```
while (!cond);
```

allora tale processo consuma inutilmente tempo di CPU.

2.2 Codice inutile

Talvolta, di fronte a dubbi sulle soluzioni da adottare, si cercano risposte da varie sorgenti: colleghi, siti web, etc. Ricopiare acriticamente frammenti di codice può, nella migliore delle ipotesi, portare ad includere linee di codice inutili. Quando si ricevono suggerimenti, si raccomanda di cercare di comprendere il codice suggerito e modificarlo secondo le proprie esigenze.

3 Consigli per semplificare debugging

Segue un elenco di consigli che potrebbero rendere più agevole lo sviluppo del progetto.

3.1 Handler Ctrl+C

Quando un programma non funziona si è soliti premere `Ctrl+C` per interrompere l'esecuzione. Nel progetto di sistemi operativi la sola pressione di `Ctrl+C`, interrompe certamente l'esecuzione del programma malfunzionante, ma **può lasciare residui di esecuzione che possono influenzare le esecuzioni seguenti!!!** Per esempio:

- se degli oggetti IPC persistenti sono stati creati, essi rimarranno allocati anche dopo la terminazione del processo. Una successiva esecuzione del processo potrebbe quindi essere influenzata dallo degli oggetti IPC già presenti (le `???get(...)` potrebbero fallire. Oppure le risorse di sistema si potrebbero saturare, etc.)l
- eventuali processi figli rimangono vivi e in esecuzione.

Per questo motivo, potrebbe essere una buona pratica quella di definire un handler di `SIGINT` (segnale inviato dalla pressione di `Ctrl+C`) che ripulisca lo stato (termini tutti i processi, elimini gli oggetti IPC, etc.)