SCANF & VALGRIND

sabato 24 dicembre 2022 15:07

SCANF = "La maggior parte dei Bug sta tra la tastiera e la sedia."

- La funzione scanf():
  - legge dei caratteri dal canale standard di input
  - li converte secondo le specifiche di formattazione fornite dalla stringa di formato
  - memorizza i valori ottenuti negli argomenti seguenti (puntatori)

ESSA È NATA PER LEGGERE FILE!

NON NASCE PER leggere dati digitati da un utente!

- Utilizzare la scanf() è complesso e può portare a molti undefined behavior

- Per la scanf() una stringa finisce dove si incontra uno spazio

ESEMPIO

```c
#include <stdio.h>

int main(void)
{
    int a;
    printf("enter a number: ");
    scanf("%d", &a);
    printf("You entered %d.\n", a);
}
```

- Eseguitelo inserendo come input: 42
- Eseguitelo inserendo come input: abcdefgh

Se inserisco un numero va tutto bene!
Se inserisco abcdefgh non scrive valori corretti.
↓
Il risultato è zero.

```c
#include <stdio.h>

int main(void)
{
    char name[12];
    printf("What's your name? ");
    scanf("%s", name);
    printf("Hello %s!\n", name);
}
```

- Eseguitelo inserendo come input: Paolo
- Eseguitelo inserendo come input: Bruno Liegi Bastonliegi
- Eseguitelo inserendo come input: Bruno-Liegi-Bastonliegi

Si ferma al primo spazio la scanf!

FFLUSH (STDIN)

Esempio 3: voglio leggere un numero

```c
#include <stdio.h>

int main(void)
{
    int a;
    printf("enter a number: ");
```

Mai usare FFLUSH().
Per stream di input associati con dei file che possono essere letti avanti e indietro, FFlush scarta tutti i dati che sono stati letti

```c
int main(void)
{
    int a;
    printf("enter a number: ");
    while (scanf("%d", &a) != 1) {
        // input was not a number, ask again:
        fflush(stdin); // <- never ever do that!
        printf("enter a number: ");
    }
    printf("You entered %d.\n", a);
}
```

che possono essere letti avanti e indietro, FFlush scarta tutti i dati che sono stati letti in buffer;

seekable file → sono dei file su disco;

STDIN è il canale di INPUT che proviene dal Terminale!

Non devo fare FFLUSH su un canale che proviene dal Terminale:

```
FFLUSH(3)                    Library Functions Manual                    FFLUSH(3)

NAME
       fflush - flush a stream

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <stdio.h>

       int fflush(FILE *stream);

DESCRIPTION
       For  output streams, fflush() forces a write of all user-space buffered data for the given out-
       put or update stream via the stream's underlying write function.

       For input streams associated with seekable files (e.g., disk files, but not pipes or termi-
       nals),  fflush() discards any buffered data that has been fetched from the underlying file, but
       has not been consumed by the application.
```

Quindi se eseguo FFLUSH su STDIN il risultato è undefined Behaviour;
il codice non è portabile, magari ci funziona anche, ma magari qualcun altro lo ricompila su un altro sistema e scoppia tutto;

INTRODUZIONE VALGRIND SU LINUX

Esempio 5: voglio leggere una stringa

file: io4.c

```c
#include <stdio.h>

int main(void)
{
    char name[40];
    printf("What's your name? ");
    scanf("%39[^\n]", name);
    printf("Hello %s!\n", name);
}
```

Acquisisci per 39 caratteri SMO a quando non premi "INVIO" → \n

Leggi da STDIN al più 39 caratteri!

- Eseguitelo con input: Bruno Liegi Bastonliegi
- Eseguitelo schiacciando solo return (eventualmente con valgrind)

A primo impatto sembra che vada tutto bene.

```
[pellegrini@X1 4. Input e Output]$ gcc io4.c
[pellegrini@X1 4. Input e Output]$ ./a.out
What's your name? Bruno Liegi Bastonliegi
Hello Bruno Liegi Bastonliegi!
```

MA L'UNDEFINED BEHAVIOUR È SEMPRE DIETRO L'ANGOLO!

>> VALGRIND ./file_eseguibile

ho dato come INPUT una stringa vuota

```
What's your name?
==4669== Conditional jump or move depends on uninitialised value(s)
==4669==    at 0x4847D09: strlen (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4669==    by 0x48EC737: __vfprintf_internal (vfprintf-process-arg.c:397)
==4669==    by 0x48E081E: printf (printf.c:33)
==4669==    by 0x1091B9: main (in /home/pellegrini/Insync/ilpelle@gmail.com/Google Drive/Documenti/Universita/Lezioni/Calcolatori Elettronici/esempi/4. Input e Output/a.out)
==4669==
Hello !
==4669==
==4669== HEAP SUMMARY:
==4669==     in use at exit: 0 bytes in 0 blocks
==4669==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==4669==
==4669== All heap blocks were freed -- no leaks are possible
==4669==
==4669== Use --track-origins=yes to see where uninitialised values come from
==4669== For lists of detected and suppressed errors, rerun with: -s
==4669== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
[pellegrini@X1 4. Input e Output]$
```

STO andando a leggere un qualche dato che NON È MAI STATO INIZIALIZZATO.

↓

SONO ENTRATO IN UNDEFINED BEHAVIOR!

Siccome la SCANF ha TROVATO SUBITO una STRINGA VUOTA (\n), NON SCRIVE NIENTE.

E quindi PRINTF, legge un buffer che NON È STATO INIZIALIZZATO!

E questo NON va bene e non ci stiamo per leggere una stringa!

ORA USO lo STESSO IDENTICO codice E AGGIUNGO UNO SPAZIO DAVANTI SULLA SCANF:

Esempio 6: voglio leggere una stringa

file: io5.c

```c
#include <stdio.h>

int main(void)
{
    char name[40];
    printf("What's your name? ");
    scanf(" %39[^\n]", name);
    //      ^ note the space here, matching any whitespace
    printf("Hello %s!\n", name);
}
```

SPAZIO

- Eseguitelo schiacciando solo return
- Eseguitelo con input vuoto (CTRL+D su Linux, eventualmente con valgrind)

Lo SPAZIO PRIMA di una STRINGA, fa il MATCH TRA UN qualsiasi SPAZIO VUOTO, anche ZERO SPAZI VUOTI!

↓

NON POSSO DARGLI SPAZI VUOTI.

ORA esiste un modo sui TERMINALI PER DIRE: "voglio chiudere il canale STDIN"

Significa mandare un carattere speciale che si chiama EOF (end of file).

CTRL + D su linux.

↳ E IL PROCESSO SI CHIUDE! MA SE ACCEDO CON valgrind, ho lo STESSO ERRORE di PRIMA.

Quindi OBBLIGANDO l'utente a SCRIVERE TUTTO TRANNE che SPAZI, comunque l'UTENTE TROVA UN MODO PER NON SCRIVERE NULLA! NEMMENO così va BENE!

ALLORA PROVIAMO A LEGGERE QUALCOSA!

NON USO più la SCANF, PERCHÉ NON SO che TIPO di dato va processato.

Esempio 7: ok, voglio leggere *qualcosa*

file: io6.c

```c
#include <stdio.h>

int main(void)
{
    char name[40];
    printf("What's your name? ");
    if (fgets(name, 40, stdin)) {
        printf("Hello %s!\n", name);
    }
}
```

- Eseguitelo con qualsiasi input

FGETS ();

Vado nel manuale Fgets(), digito prima:

>> MAN STDIO

```
simoneremoli — less - man stdio — 110×29
ferror      check and reset stream status
fflush      flush a stream
fgetc       get next character or word from input stream
fgetln      get a line from a stream
fgetpos     reposition a stream
fgets       get a line from a stream
fgetwc      get next wide character from input stream
fgetws      get a line of wide characters from a stream
fileno      check and reset stream status
fopen       stream open functions
fprintf     formatted output conversion
fpurge      flush a stream
fputc       output a character or word to a stream
fputs       output a line to a stream
fputwc      output a wide character to a stream
```

Mi permette di leggere da un determinato

Mi permette di leggere da un determinato file, verso un determinato BUFFER, un certo numero di caratteri! ↳ name

```
filenc      check and reset stream status
fopen       stream open functions
fprintf     formatted output conversion
fpurge      flush a stream
fputc       output a character or word to a stream
fputs       output a line to a stream
fputwc      output a wide character to a stream
fputws      output a line of wide characters to a stream
fread       binary stream input/output
freopen     stream open functions
fropen      open a stream
fscanf      input format conversion
fseek       reposition a stream
fseeko      reposition a stream
ftell       reposition a stream
funopen     open a stream
fwide       set/get orientation of stream
fwopen      open a stream
fwprintf    formatted wide character output conversion
fwrite      binary stream input/output
```

LEGGIAMO IL MANUALE DI FGETS().

Fgets() legge al più un carattere IN MENO di size! (perchè mette sempre \0)

```
cast to an int, or EOF on end of file or error.

getc() is equivalent to fgetc() except that it may be implemented as a macro
which evaluates stream more than once.

getchar() is equivalent to getc(stdin).

fgets() reads in at most one less than size characters from stream and stores
them into the buffer pointed to by s.  Reading stops after an EOF or a newline.
If a newline is read, it is stored into the buffer.  A terminating null byte
('\0') is stored after the last character in the buffer.

ungetc() pushes c back to stream, cast to unsigned char, where it is available
for subsequent read operations.  Pushed-back characters will be returned in re-
verse order; only one pushback is guaranteed.

Calls to the functions described here can be mixed with each other and with
calls to other input functions from the stdio library for the same input stream.

For nonlocking counterparts, see unlocked_stdio(3).

RETURN VALUE
    fgetc(), getc(), and getchar() return the character read as an unsigned char
```

perchè poi legge IL TERMINATORE \0!
La lettera si sposta dopo un "a capo" = \n
Quindi legge alla grana della linea:
Ogni volta che premo INVIO, vengono consegnati questi dati!

Se viene letto '\n' viene saltato!

GESTISCO L'ACCAPO:

Esempio 8: ci siamo!

file: io7.c

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char name[40];
    printf("What's your name? ");
    if (fgets(name, 40, stdin)) {
        name[strcspn(name, "\n")] = 0;
        printf("Hello %s!\n", name);
    }
}
```

È STATO INSERITO UN CARATTERE DI A CAPO?

→ TROVO IL PUNTO IN CUI L'ACCAPO È STATO INSERITO!
STRCSPN mi dà la posizione e al suo posto ci metto IL TERMINATORE

- Eseguitelo con qualsiasi input

OUTPUT.

```
What's your name? 7565e65e
Hello 7565e65e!
```

effettivamente \n viene tolto!

ORA PERÒ VORREI leggere un numero:
Mi PIACE considerare IL NUMERO come un WORD.

Esempio 9: come faccio a leggere un numero?

file: io8.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a;
    char buf[1024]; // use 1KiB just to be sure

    do {
        printf("enter a number: ");
        if (!fgets(buf, 1024, stdin)) {
            // reading input failed, give up:
            return 1;
        }

        // have some input, convert it to integer:
        a = atoi(buf);
    } while (a == 0); // repeat until we got a valid number

    printf("You entered %d.\n", a);
}
```


ATOI = ALPHA NUMERIC TO INTEGER

RETURN 0 IN CASO di ERRORI

RETURN LA CONVERSIONE IN CASO di SUCCESS.

```
ATOI(3)                    Library Functions Manual                    ATOI(3)

NAME
       atoi, atol, atoll - convert a string to an integer

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <stdlib.h>

       int atoi(const char *nptr);
       long atol(const char *nptr);
       long long atoll(const char *nptr);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       atoll():
           _ISOC99_SOURCE
               || /* Glibc <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION
       The  atoi()  function  converts  the initial portion of the string pointed to by
       nptr to int.  The behavior is the same as

           strtol(nptr, NULL, 10);
 Manual page atoi(3) line 1 (press h for help or q to quit)
```

fin tanto che a è uguale a 0, RIPETI!

IO DEVO SAPERE A PRIORI CHE CIÒ CHE INSERISCO È UN NUMERO!

file: io9.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main(void)
{
    long a;
    char buf[1024]; // use 1KiB just to be sure
    int success; // flag for successful conversion

    do {
        printf("enter a number: ");
        if (!fgets(buf, 1024, stdin)) {
            // reading input failed:
            return 1;
        }

        // have some input, convert it to integer:
        char *endptr;

        errno = 0; // reset error number
        a = strtol(buf, &endptr, 10);
        if (errno == ERANGE) {
            printf("Sorry, this number is too small or too large.\n");
            success = 0;
        } else if (endptr == buf) {
            // no character was read
            success = 0;
        } else if (*endptr && *endptr != '\n') {
            // *endptr is neither end of string nor newline,
            // so we didn't convert the *whole* input
            success = 0;
        } else {
            success = 1;
        }
    } while (!success); // repeat until we got a valid number

    printf("You entered %ld.\n", a);
}
```

STRTOL, ho bisogno di un puntatore aggiuntivo chiamato endpointer

↳ se endptr non è NULL STRTOL salva l'indirizzo del primo carattere non valido in endpointer.

↓

se trova qualcosa che non è un carattere numerico, INTERROMPE la conversione e scrive dentro il puntatore L'INDIRIZZO DEL PRIMO CARATTERE CHE NON È RIUSCITO A CONVERTIRE!

non ha processato tutti quanti i caratteri

Se non c'erano cifre, viene restituito il valore originale in endPointer e viene restituito return 0.

Il valore di ritorno è il risultato della conversione!!!

Se si verifica un errore di conversione, errno viene impostato a ERANGE.

Se il primo carattere non valido è il primo carattere del mio buffer, allora significa che nessun carattere è stato letto!

ed è il secondo IF!

oppure endptr è arrivata alla fine della stringa letta?
cioè ho processato tutti i valori che sono stati scritti?

SCANF

Le regole di uso della scanf()

- Regola 1: `scanf()` non serve a leggere input, ma ad effettuarne un'interpretazione di una stringa.
- Regola 2: `scanf()` può essere pericolosa se utilizzata in maniera incauta. Utilizzata con le stringhe ed una lunghezza di buffer fissa è il suo uso migliore.
- Regola 3: Anche se le stringhe di formato della `scanf()` assomigliano molto a quelle della `printf()`, hanno una semantica leggermente differente (leggete il manuale!)
- Regola 4: Anche con stringhe a dimensione fissa, è possibile che vengano lasciati dei caratteri su stdin. Se questo è un problema, conviene utilizzare `fgets()`.

FGetch() legge un solo carattere alla volta da STDIN! Finché non viene individuato EOF.