PROBLEM SOLVING CON LINUX

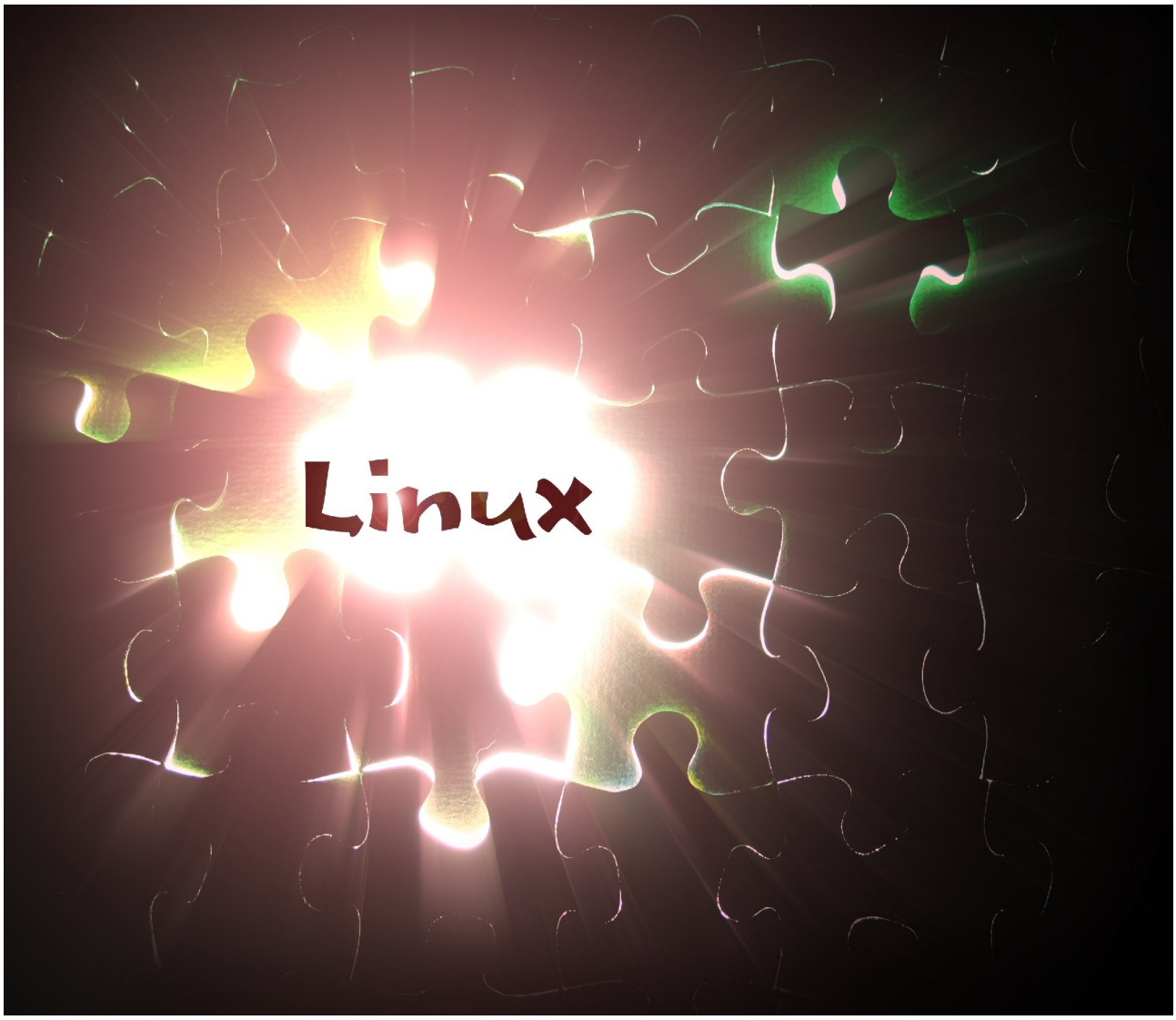


Foto: modified work downloaded on sxc.hu (author scyza http://www.sxc.hu/profile/Scyza)

ALESSIO PORCACCHIA

Alessio Porcacchia

Alessio Porcacchia appartiene alla comunità attiva opensource dal 1996. Ha lavorato e continua a lavorare come consulente per Unix-Linux e tutti i derivati SYS5 da circa 7 anni per le maggiori multinazionali del settore IT. Ha pubblicato inoltre (guida sun solaris per per il sistemista linux junior) e la documentazione per l'installazione del tool di monitoring (cricket). www.porcacchia.altervista.org

Prefazione: dedico questo articolo al blog http://3v1n0.tuxfamily.org ringraziando l'autore per aver dato un così positivo commento a l'umile lavoro di divulgazione e di supporto all'interno della comunità. Dedico inoltre alla comunità Debian Italiana tutto il lavoro svolto in questi anni. Questo articolo nacque inizialmente per una pubblicazione da rivista. Devo dire che presentai tale articolo a molte riviste e come risultato ebbi una mancato riscontro. In definitiva ritengo che un articolo sul problem solving sia cosa buona e utile, dato che manca effettivamente un qualsiasi tipo di riferimento manualistico in Italia. Questo ci fa ancora una volta capire il livello di arretratezza che ci si trova ad affrontare dal punto di vista di divulgazione cartacea IT in italia. In questi anni comincio a farmi una visione sempre piu' limpida del perchè manchi una rivista in italiano come Samag.com Ovvero una rivista dedicata dai tecnici per i tecnici, poiché almeno dal punto di vista editoriale si guarda a linux come un "esotico" prodotto per vendere qualche iso masterizzata e non come effettivo strumento di lavoro all'interno dei datacenter o dei CED, cosa che posso testimoniare direttamente comincia a diventare una realtà tangibile. Se vediamo il mercato editoriale, di riviste in italia che lontanamente solo si avvicinano a samag.com non ci sono, e non credo perchè non ci sia mercato, ma per la poca lungimiranza di certe case editrici. Si pubblicano migliaia di riviste clone per le console giochi che alla fine sono sempre tutte uguali, ma nessuno dico NESSUNO, hai mai voluto riflettere sulla possibilità di una reale rivista informatica per gli addetti hai lavori. Sinceramente questa cosa è di una tristezza sconfinata. Speriamo che in un prossimo futuro ci sia chi in Italia voglia davvero fare una rivista tecnica per i tecnici e non "materiale cartaceo" aggiuntivo al dvd.

PROBLEM SOLVING SU LINUX

E' molto comune per chi amministra una macchina linux dover affrontare il più delle volte delle problematiche assai serie sulle macchine che gestisce, chi è nel settore sfortunatamente lo sa MOLTO BENE.
in questo articolo cercherò di spiegare in modo semplice, per quanto mi e' possibile, quali sono i passi principali affinché si riesca in buona parte dei casi ad affrontare delle difficoltà di natura tecnica sui sistemi linux ed uscirne “indenni”.
Come molti sapranno, affrontare il problem solving senza una adeguata esperienza risulta assai difficoltoso e non è detto che possano bastare le esperienze acquisite durante gli anni. Cerchiamo quindi di affrontare il problema in modo diretto seguendo alcuni semplici passi per addentrarci in questo mondo irto di insidie. Ovviamente questo tipo di situazione non può essere globalmente incluso in un singolo articolo, però può dare al lettore una visione d'insieme dando e un buon punto di partenza per poter iniziare ad analizzare in modo ordinato tali problematiche. Noi cercheremo in questo articolo di dedicarci specificatamente in questo articolo del problem solving a livello sistema-applicativo

Cominciamo col dire che il tipo di problemi che generalmente a livello sistema-applicativo si affrontano possono essere suddivise in questo modo:

Come possiamo evincere dal grafico il problema che si affronta maggiormente e' di tipo di performance, che si intende tutto quello che rallenta fisicamente il programma (colli di troughtput , problemi di memoria, etc) in seconda istanza il

problema principale e l'uscita inaspettata del programma, infine l'hang e i problemi riscontrati a livello di kernel panic.
i tool che vanno presi in considerazione sono :

- ***TOP (lista dei processi di default che usano maggiormente la cpu)***
- ***LSOF (tool che da una lista dei file aperti da un determinato programma con il loro ID e i file descriptors***
- ***STRACE tool di tracciamento delle system calls***
- ***LTRACE simile a strace***
- ***GDB un debugger molto potente***

Prima di iniziare in modo approfondito ad affrontare il problema che abbiamo riscontrato sulla nostra macchina applichiamo inizialmente alcune principali regole e rispondiamo alle seguenti domande:

- il problema che abbiamo riscontrato e' già capitato nelle nostre precedenti esperienze?

- che tipo di importanza e impatto ha sulla macchina/e che stiamo amministrando (definire la priorità di soluzione e il conseguente tempo di disservizio che può generare)

- Abbiamo informazioni sufficienti e complete a riguardo del problema che abbiamo riscontrato?

- sono avvenute in tempi recenti delle modifiche sul sistema prima che avvenisse il problema? (tuning del kernel, modifiche hardware/software etc)

- il programma o l'applicativo ha avuto un maggior aumento di carico di lavoro di recente? (appesantimento di carico dell'applicativo)

Queste domande sicuramente aiutato in modo iniziale a comprende quale sia il modo migliore per cominciare ad affrontare il problema che abbiamo riscontrato. Dopo aver affrontato queste domande iniziali ci possiamo affidare a internet, per vedere se esistono all'interno di mailing list, Motori di Ricerca, manualistica on-line su linux. A questo punto seguire come una checklist i seguenti passi:

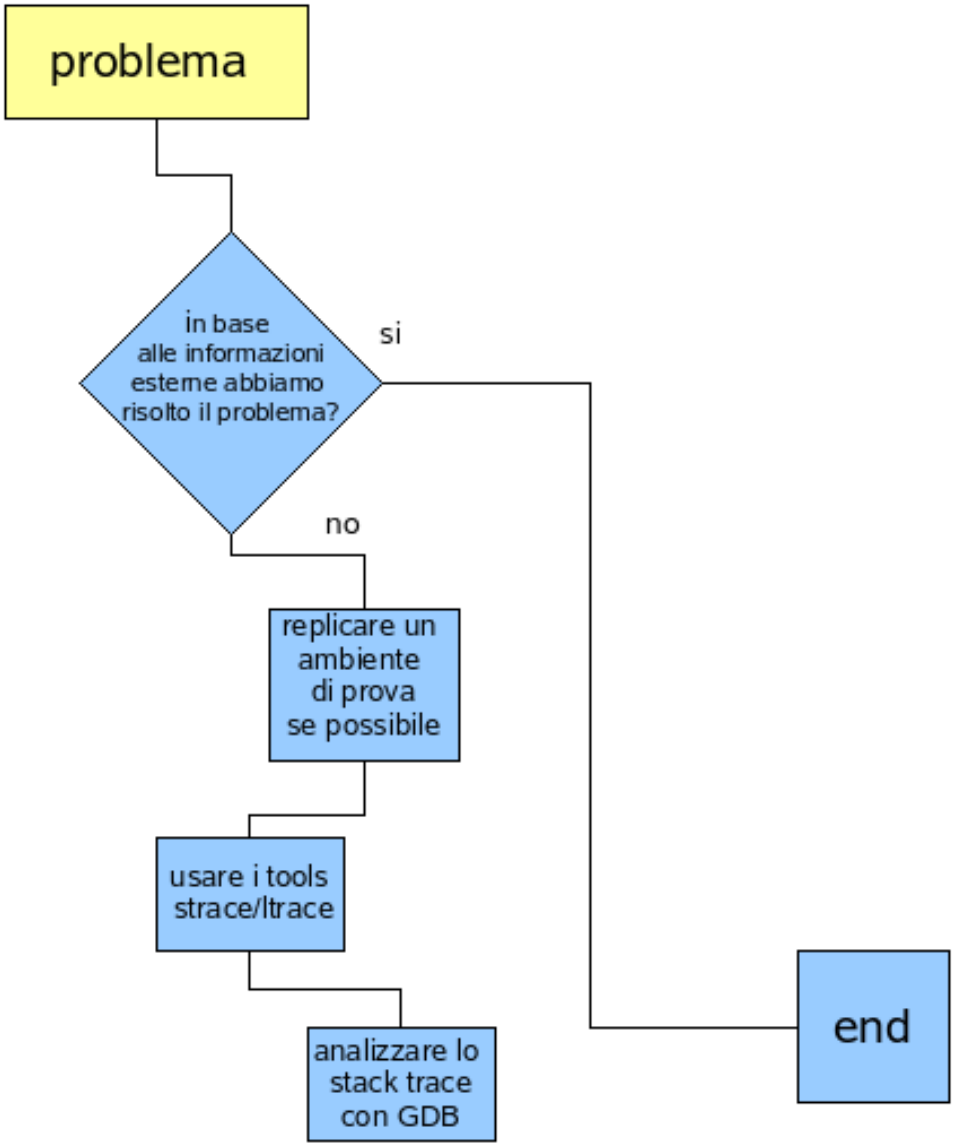
1. analizzare il maggior numero di informazioni inerenti al problema usando tutti i log disponibili e cercare di ricostruire anche con uno storico tutto quello che è avvenuto sulla macchina.

2. risalire all'origine del problema escludendo tutto quello che non è inerente al problema

3. ove fosse possibile cercare di ricreare un ambiente di test idoneo a quello dove e' avvenuto il problema cercando di replicare nel modo più possibile quello dove si e' riscontrato il problema

4. analizzare il codice sorgente del programma per controllare se non ci sia qualche bug nel codice.

Definito questo cominciamo a seguire ad approfondire la problematica da un punto di vista tecnico:



le possibili cause di un crash dell'applicativo può essere rintracciato nel Trap di un processo.

IL TRAP IN UN PROCESSO.
Quando il kernel non riesce a mandare in esecuzione un programma invia un segnale. Un segnale e' un interruzione software che notifica un evento signal() . Linux ad esempio ne definisce 32. e quasi tutti sono rintracciabili sotto signal.h per dare un occhiata ai vari segnali (man sigaction). Noi prenderemo in esame solo alcuni di questi segnali. un processo quando riceve uno di questi segnali reagisce in tre modi: gestisce il segnale in base ad una funzione di routine handler definita dal

programmatore , esegue una funzione predefinita dal sistema operativo (funzione di default) o ignora il segnale.

Noi prenderemo in considerazione solo alcuni segnali :

- SIGSEGV
- SIGBUS
- SIGILL
- SIGABRT
- SIGTRAP
- SIGSYS
- SIGFPE

SIGSEV: (SEGMENT VIOLATION) il programma sta cercando di leggere o scrivere in una zona di memoria protetta al di fuori di quella che gli è stata riservata dal sistema. A generare il segnale è il meccanismo di protezione della memoria che si accorge dell'errore. da cui possono definirsi i seguenti errori:

- overrun di memoria
- errore “out of memory”
- variabile o pointer non inizializzato

SIGBUS: (BUS ERROR) e' un segnale che viene generato di solito quando si dereferenzia un puntatore non inizializzato, sta a indicare un accesso non permesso su un indirizzo esistente (out heap or stack) indica l'accesso ad un indirizzo non valido.

- puntatore Null non dereferenziato
- Paging Error (page fault)

SIGILL:(ILLEGAL ISTRUCTION) il programma sta cercando di eseguire una istruzione privilegiata o inesistente, cerca di eseguire codice non lecito e indica che l'ELF è corrotto . Un esempio classico e quando viene generato uno Stack Overflow.

- Corruzione dello Stack
- mancato caricamento delle shared libraries
- ELF corrotto

SIGABRT: (ABORT) e' segnale indica che il programma stesso ha rilevato un errore che viene riportato chiamando la funzione abort()

SIGTRAP: (TRAP) è un segnale generato da un'istruzione di breakpoint o dall'attivazione del tracciamento del processo. E' usato nei programmi di Debugging

- codice scritto in modo errato

SIGSYS:(SYSTEM CALL) Sta ad indicare che si è eseguita una istruzione che richiede l'esecuzione di una system call, e' stato fornito un codice sbagliato per quest'ultima.

- codice non corretto

SIGFPE: (FLOATING POINT EXECPTION) è un errore di tipo aritmetico compresa la divisione per zero e l'overflow.

- errori di calcolo nella divisione per zero
- overflow

Abbiamo in oltre anche eventi molto più gravi e generalizzati che rientrano del Kernel Panic, analizzare in modo approfondito la creazione di un panic Kernel e generalmente molto più difficoltoso e può risultare senza una adeguata conoscenza del codice di difficilissima risoluzione.

L'HANG E IL RALLENTAMENTO DI UN PROCESSO

L'hang di un applicativo o un serio rallentamento di un processo avviene in due casi:

1. quando tale processo a un consumo di risorse per la CPU quasi nullo (possibile controllo grazie a tools come top o htop) in questo caso viene definito (stuck of system call) quando una system call attende all'infinito un evento che non avviene.
2. quando il carico di consumo di CPU del processo diventa molto pesante e in questo caso è sempre dovuto a una ripetizione in loop di codice (code spinning).

uno stuck avviene quando :

- e' atteso un semaforo (IPC inter-process comunication sincronizzazione di processi)
- e' atteso un evento a livello di rete o di Filesystem condiviso (esempio NFS)
- e' atteso un qualsiasi evento che non avviene

il code spinning invece avviene per i seguenti motivi :

- codice errato che entra in loop
- un loop di tipo TIMEOUT
- un tipo qualsiasi di loop

un hang può anche avvenire avviene per un latch di un'applicazione di multiprocesso, un latch avviene quando i vari processi che appartengono allo stesso

stack aspettano che una risorsa sia rilasciata mentre tale risorsa rimate lockata da il processo originario che non rilascia la risorsa. (deadlock) Infine ovviamente un hang avviene quando il sistema e sovraccarico da altri programmi e non riesce a gestire in modo adeguato il processo, quest'ultimo avviene quando l'hardware e' sotto dimensionato o quando ci sono dei colli di bottiglia o dei malfunzionamenti a livello hardware, quindi fate molta attenzione allo stato hardware della macchina.

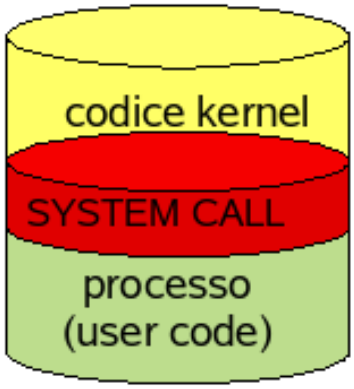
UTILIZZARE STRACE/LTRACE

A questo punto e' duopo dopo aver analizzato il tipo da cosa sia prodotto il problema e si e' compreso che potrebbe essere dovuto a codice non corretto utilizzare due tool per il debug fisico dell'applicativo. strace e ltrace.

Strace e' un tool che server per poter analizzare e tracciare le chiamate di sistema, (system call) e risulta un ottimo tool di investigazione. Prima di passare alla parte pratica definiamo cosa e' una system call.

LA SYSTEM CALL

Una system call o chiamata di sistema e uno strato tra Il codice Kernel (Kernel Code) e processo (user code):



Potremmo definire in modo adeguato una system call come una funzione che risiede ed e' eseguita nel kernel, provvede ad un adeguata e sicura gestione delle risorse (dischi, memoria e rete) all'accesso dei servizi del kernel (inter-process comunication e alle informazioni del sistema) e richiede per poter funzionare correttamente tre principali tipi di istruzioni (gate, trap, interrupt) comprendere approfonditamente come lavora una system call è fondamentale per avere una visione approfondita per riuscire ad analizzare approfonditamente il problema riscontrato.

Per riuscire a capire cosa avviene e come lavora una syscall. Ovviamente una buona conoscenza dell'assembler sicuramente tende a dare una visione migliore di ciò che accade. Prendiamo ad esempio la system call read:

```
readelf -sd /lib/libc.so.6 | grep read
```

690: 000b3860 239 FUNC WEAK DEFAULT 11 pread64@@GLIBC_2.1

698: 000d02d0 45 FUNC GLOBAL DEFAULT 11 pthread_attr_setdetachsta@@GLIBC_2.0

818: 0005b880 86 FUNC GLOBAL DEFAULT 11 fread_unlocked@@GLIBC_2.1

883: 000b5ca0 118 FUNC WEAK DEFAULT 11 read@@GLIBC_2.0 <-----

898: 000d7400 52 FUNC GLOBAL DEFAULT 11 __pread_chk@@GLIBC_2.4

Ora vediamo in modo piu' approfondito a livello di codice assembler cosa accade:

objdump -d /lib/libc.so.6 |less

b5ca0:000b5ca0 <__read>:

65 83 3d 0c 00 00 00 cmpl $0x0,%gs:0xc

b5ca7: 00

b5ca8: 75 1d jne b5cc7 <__read+0x27>

b5caa: 53 push %ebx

b5cab: 8b 54 24 10 mov 0x10(%esp),%edx

b5caf: 8b 4c 24 0c mov 0xc(%esp),%ecx

b5cb3: 8b 5c 24 08 mov 0x8(%esp),%ebx

b5cb7: b8 03 00 00 00 mov $0x3,%eax <----------

b5cbc: cd 80 int $0x80 <----------

b5cbe: 5b pop %ebx

b5cbf: 3d 01 f0 ff ff cmp $0xfffff001,%eax

b5cc4: 73 2d jae b5cf3 <__read+0x53>

b5cc6: c3 ret

b5cc7: e8 24 ac 01 00 call d08f0 <pthread_exit+0xe0>

b5ccc: 50 push %eax

b5ccd: 53 push %ebx

b5cce: 8b 54 24 14 mov 0x14(%esp),%edx

b5cd2: 8b 4c 24 10 mov 0x10(%esp),%ecx

b5cd6: 8b 5c 24 0c mov 0xc(%esp),%ebx

b5cda: b8 03 00 00 00 mov $0x3,%eax <----

```
b5cdf:    cd 80            int   $0x80  <------------

b5ce1:    5b             pop   %ebx

b5ce2:    87 04 24         xchg  %eax,(%esp)

b5ce5:    e8 c6 ab 01 00      call  d08b0 <pthread_exit+0xa0>

b5cea:    58             pop   %eax
```

Come avrete notato abbiamo preso per esempio l'esecuzione read come ho indicato sopra l'istruzione mov precede sempre l''istruzione di interrupt *int 0x80* in questo caso l'istruzione mov “muove” la chiamata di sistema 3 nel registro eax e l'istruzione di interrupt switcha l'esecuzione di tale thread sul kernel. . In pratica quando vediamo questo tipo di istruzione ***int 0x80*** c'e' sempre un invocazione di una system call. l'argomento di una system call viene registrato nel registro EAX. I parametri delle chiamate ed impiegano i registri di cpu in ordine canonico (eax,ebx,ecx,edx,...). Il valore di ritorno e' nel registro eax. quindi potremmo altresì rappresentare na syscall come:

EAX = syscall(EBX, ECX, EDX, ESI, EDI)

Non addentrandoci ulteriormente nell'assembler e avendo compreso in linea generale come funziona una syscall vediamo da vicino l'utilizzo del tool strace. crieiamo un seplice programmino che contiene la system call open()

```
//open.c

//esempio di system call

//  per vedere come funziona strace

//

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main( )

{

int fd ;

int i = 0 ;

fd = open( "/home/elwood/nonesisto", O_RDONLY ) ;

if ( fd < 0 )

i=5;

else
```

```
i=2;

return i;

}
```

compiliamolo con il gcc:

gcc open.c -o open.o

e a questo punto proviamo ad utilizzare strace per comprendere come funziona tale tool e lanciamolo proprio su il nostro programma che abbiamo creato:

```
strace ./open.o

execve("./open.o", ["./open.o"], [/* 66 vars */]) = 0

brk(0)                          = 0x804a000

mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7f94000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)

open("/etc/ld.so.cache", O_RDONLY)      = 3

fstat64(3, {st_mode=S_IFREG|0644, st_size=90013, ...}) = 0

mmap2(NULL, 90013, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7f7e000

close(3)                        = 0

open("/lib/i686/libc.so.6", O_RDONLY)   = 3

read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320^\1"..., 512) = 512

fstat64(3, {st_mode=S_IFREG|0644, st_size=1232576, ...}) = 0

mmap2(NULL, 1238404, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb7e4f000

mmap2(0xb7f78000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x129) = 0xb7f78000

mmap2(0xb7f7b000, 9604, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7f7b000

close(3)                        = 0

mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7e4e000

set_thread_area({entry_number:-1 -> 6, base_addr:0xb7e4e6c0, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0

mprotect(0xb7f78000, 4096, PROT_READ)   = 0
```

mprotect(0xb7fad000, 4096, PROT_READ) = 0

munmap(0xb7f7e000, 90013) = 0

open("/home/elwood/nonesisto", O_RDONLY) = -1 ENOENT (No such file or directory) <- - -

exit_group(5) = ?

Process 21050 detached

[elwood@localhost ~]$

come noteremo nella riga indicata si tenta di aprire con una system call open un file appunto (/home/elwood/nonesisto) rendendo un valore di tipo negativo e l'uscita da parte del programma eseguito.

ma definiamo per ordine cosa accade quando abbiamo lanciato uno strace:

execve: e la prima cosa che vediamo ed essa sta a indicare che e' stato evocata una system call exec() questa appare quando viene lanciato un nuovo processo tale system call come vedete indica chiaramente il path del programma che è stato eseguito e che a creato un nuovo processo. in questo caso open.o

BRK: la system call brk serve per allocale o liberare memoria dinamica (memory management per il processo)

mmap2: questo sta a indicare che sta mappando in memoria una pagina in questo caso sta creando una pagina di memoria e si trova in 0xb7f94000 nel nostro caso come potete vedere può essere per essere messo in protezione scrittura e in lettura controllando l'ultima parte della stringa vedere se c'e' una protezione di scrittura o lettura della page (pagina) di memoria.

access: qui sta ad indicare un controllo di permessi sulla ld.so.preload questo file contiene una lista delle Shared Libraries che vanno caricate prima che programma possa partire.

open: quando viene aperto un file in questo caso si ha a che fare con ld.so.cache per intenderci quando si usa ldconfig si ha che fare proprio con questo file che serve per velocizzare il caricamento delle librerie.

close:system call che indica una terminazione o chiusura di un processo

read: in questo caso sta a indicare un caricamento dell' ELF header per la libreria c

fstat64: riguarda controlla lo status information del file indicato nel descrittore

mprotect: rimuove qualsiasi permesso per la regione di memoria utilizzata
0xb7f78000 quella che come avrete notato era stata definita da mmap2

nunmap: cancella la porzione di memoria per uno specificato adress range e per l'appunto la porzione di memoria che era stata definita inizialmente da mmap2

la penultima riga: come abbiamo potuto vedere noi nel programma abbiamo indicato l'apertura di un file inesistente “/home/elwood/nonesisto” l'errore ENOENT (-1 quando e' negativo indica che c'e' un errore) in questo caso sta a indicare che un file oppure una delle librerie dinamiche non esistono come abbiamo potuto vedere, indica fisicamente un output di errore classico (no such file or directory).
exit: system call che indica lo status di uscita del programma.

Molto importanti sono anche le opzioni che vengono date al comando strace vediamone alcune:
la prima che è sicuramente utile per il problem solving e con l'opzione -f che serve per tracciare anche i processi figli prendiamo ad esempio una parte di strace di proftpd:

strace -f /usr/sbin/proftpd

```
setresgid32(-1, 65534, -1)              = 0
setresuid32(-1, 65534, -1)              = 0
rt_sigprocmask(SIG_UNBLOCK, [HUP INT QUIT BUS USR1 ALRM TERM CHLD IO], NULL, 8) = 0
clone(Process 21447 attached
child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0xb7d74708) = 21447
[pid 21446] exit_group(0)               = ?
[pid 21447] setsid(Process 21446 detached
)                 = 21447
close(0)                          = 0
close(1)                          = 0
close(2)                          = 0
setpgid(0, 21447)                       = -1 EPERM (Operation not permitted)
chdir("/")                        = 0
lstat64("/", {st_mode=S_IFDIR|0555, st_size=4096, ...}) = 0
rt_sigprocmask(SIG_BLOCK, [HUP INT QUIT BUS USR1 ALRM TERM CHLD IO], NULL, 8) = 0
setresuid32(-1, 0, -1)                  = 0
setresgid32(-1, 0, -1)                  = 0
rt_sigprocmask(SIG_UNBLOCK, [HUP INT QUIT BUS USR1 ALRM TERM CHLD IO], NULL, 8) = 0
stat64("/var/run/proftpd/proftpd.scoreboard", {st_mode=S_IFREG|0644, st_size=16, ...}) = 0
unlink("/var/run/proftpd/proftpd.scoreboard") = 0
open("/var/run/proftpd/proftpd.scoreboard", O_RDWR|O_CREAT|O_LARGEFILE, 0644) = 0
fchmod(0, 0644)                         = 0
fstat64(0, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
rt_sigprocmask(SIG_BLOCK, [HUP INT QUIT BUS USR1 ALRM TERM CHLD IO], NULL, 8) = 0
read(0, "", 16)                         = 0
rt_sigprocmask(SIG_UNBLOCK, [HUP INT QUIT BUS USR1 ALRM TERM CHLD IO], NULL, 8) = 0
time(NULL)                              = 1171295584
fcntl64(0, F_SETLKW64, {type=F_WRLCK, whence=SEEK_SET, start=0, len=0}, 0xbfb8f0a0) = 0
write(0, "\357\276\255\336\2\0\4\1\307S\0\0`\215\320E", 16) = 16
fcntl64(0, F_SETLK64, {type=F_UNLCK, whence=SEEK_SET, start=0, len=0}, 0xbfb8f0a4) = 0
rt_sigprocmask(SIG_BLOCK, [HUP INT QUIT BUS USR1 ALRM TERM CHLD IO], NULL, 8) = 0
geteuid32()                             = 0
setresgid32(-1, 65534, -1)              = 0
setresuid32(-1, 65534, -1)              = 0
```

come avrete potuto notare con questa opzione si può tracciare con i differenti pid i processi figli creati.
Con l'opzione -tt viene immesso anche il timestamp compresi i millisecondi :
strace -tt ./open.o

....
16:58:00.651833 *execve("./open.o", ["./open.o"], [/* 50 vars */]) = 0*
16:58:00.652223 *brk(0) = 0x804a000*
16:58:00.652333 *mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7f12000*
16:58:00.652433 *access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)*
16:58:00.652545 *open("/etc/ld.so.cache", O_RDONLY) = 3*
16:58:00.652678 *fstat64(3, {st_mode=S_IFREG|0644, st_size=90013, ...}) = 0*
16:58:00.652795 *mmap2(NULL, 90013, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7efc000*
16:58:00.652882 *close(3) = 0*

L'opzione -T maiuscola può risultare altrettanto interessante poiché indica utilizzando le parentesi <> quanti secondi e microsecondi ha eseguito tale la syscall indicata:

mmap2(NULL, 90013, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7f68000 ***<0.000008>***

infine sicuramente una delle più interessanti l'opzione -c che riassume in tabella il debug eseguito sul nostro open.o

```
[elwood@localhost Linux]$ strace -c ./open.o
Process 21475 detached
%time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  nan    0.000000           0         1           read
  nan    0.000000           0         3         1 open
  nan    0.000000           0         2           close
  nan    0.000000           0         1           execve
  nan    0.000000           0         1         1 access
  nan    0.000000           0         1           brk
  nan    0.000000           0         1           munmap
  nan    0.000000           0         2           mprotect
  nan    0.000000           0         6           mmap2
  nan    0.000000           0         2           fstat64
  nan    0.000000           0         1           set_thread_area
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    21         2 total
```

LTRACE
e un tool che traccia intercetta e registra le chiamate delle librerie dinamiche effettuate dal processo oltre che ovviamente le chiamate di sistema:

```
[elwood@localhost Linux]$ ltrace ./open.o
__libc_start_main(0x8048374, 1, 0xbfc54cb4, 0x80483e0, 0x80483d0 <unfinished ...>
open("/home/elwood/nonesisto", 0, 027761246010)                    = -1
+++ exited (status 5) +++
```

altri tool che possono tornare utili per gestire un problem solving sono:

- GDB: GNU project command line debugger.v iene usato per fare un debug ad un processo usando la CLI debugger
- PS: utilizzato per listare i processi in running sul sistema
- TOP: lista i processi con maggiore utilizzo di cpu

- PSTREE: visualizza un albero di processi e la radice viene gestita in base ai pid e indicando i processi figli

In conclusione possiamo effettivamente dire che dopo aver seguito tali passi vi sono buone possibilità di risolvere il problema che abbiamo riscontrato. Certo non sempre il sistemista può mettersi a fare code debugging, ma se i workaround sul programma o prodotto installato non bastano alla fine vale sempre la regola che per risolvere i problemi non c'e' miglior tecnico di noi stessi.