ptrace(), exec()

Usando le due syscall ptrace ed exec vediamo come e' possibile "implementare" un semplice "quasi-debugger".

```
int execlp(const char *file, const char *arg, ...);
```

"rimpiazza" l'attuale immagine del processo con quella dell'eseguibile specificato dalla stringa "file".

```
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

ptrace() permette ad un processo di osservare e controllare l'esecuzione di un altro processo.

- request e' il tipo di azione che si vuole intraprendere
- pid e' chiaramente il "process ID" del processo che si intende tracciare
- gli altri argomenti dipendono dal tipo di azione specificata mediante il parametro request

ptrace() in azione

Vediamo come e' possibile ad esempio monitorare l'esecuzione di un semplice processo target (esempio directory 1).

```
$ make
gcc -o target target.c
gcc -DFIRSTT -o tracer1 tracer_mine.c
gcc -o tracer2 tracer_mine.c
$ ./tracer1 $PWD/target
val: DEADBEAF
$ ./tracer2 $PWD/target
Here I am 1
```

gdb

Per poter usare gdb per il debugging di un programma, e' “necessario” compilare lo stesso con l'opzione “-g”. Vediamo le differenze nel caso in cui venga usata o meno l'opzione '-g':

```
$ gcc -o main main.c
$ ls -altr main
-rwxr-xr-x 1 redo thch 6836 May 12 11:12 main
$ strings -a main  > without
$ gcc -g -o main main.c $ ls -altr main
-rwxr-xr-x 1 redo thch 9460 May 12 11:12 main
$ strings -a main > with
$ diff with without
...
```

gdb

E' possibile usare strip per eliminare i simboli “extra” dal nostro file ELF eseguibile:

```
$ strip -g main
```

Cenni sull'uso di gdb:

```
$ gdb ./main
...
(gdb) r
Starting program: /home/redo/Lezioni/Inform/lab_gen_inf/2008/slides/8/main
Hello!
^X
Program received signal SIGINT, Interrupt.
```

gdb

```
main () at main.c:11
11          }
(gdb) where
#0  main () at main.c:11
(gdb) list
6
7           while (1) {
8             int i;
9
10            i++;
11          }
12
13          return 0;
14        }
```

gdb

Proviamo dunque ad eliminare i simboli di debugging:

```
$ strip -g main
$ gdb ./main
...
(gdb) r
Starting program: /home/redo/Lezioni/Inform/lab_gen_inf/2008/slides/8/main
(no debugging symbols found)
(no debugging symbols found)
Hello!
^X
Program received signal SIGINT, Interrupt.
0x00000000004004ff in main ()
(gdb) where
#0  0x00000000004004ff in main ()
```

strace

Strace permette di “tracciare” tutte le syscall e i segnali:

```
$ strace main1
execve("./main1", ["main1"], [/* 54 vars */]) = 0
brk(0)                                  = 0x601000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2aaaaaaab000
uname({sys="Linux", node="banquo.thch.unipg.it", ...}) = 0
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/opt/intel/fce/9.1.041/lib/tls/x86_64/libc.so.6", O_RDONLY) = -1 ENOENT (No such file or directory)

...
```

vedete anche “ltrace” (traccia le chiamate alle librerie)

Profiler

Passiamo adesso ai profilers. Un profiler ci permette di analizzare con un certo dettaglio un programma allo scopo eventuale di migliorarne le prestazioni e non solo (N.B. I profiler devono essere sempre usati con “cautela” e “senno”).

Prima di passare all'uso di un profiler il semplice uso di “time” puo' di per se darci molte informazioni utili:

```
$ gcc -o main1_print main1_print.c
$ time ./main1_print
...
real    0m0.672s
user    0m0.060s
sys     0m0.103s
```

Profiler

In questo semplice esempio un banale miglioramento dell'I/O e' sufficicente a ridurre drasticamente i tempi di esecuzione:

```
$ time ./main1_print > out
```

```
real    0m0.048s
user    0m0.046s
sys     0m0.002s
```

gprof

Una volta “eliminato” l'ovvio l'uso di un profile puo' risultare indispensabile al miglioramento delle performances di un programma.

```
$ gcc -pg -o main1 main1.c
$ ./main1
Hello!
F1
F2
$ ls -altr gmon.out
-rw-r--r-- 1 redo thch 532 May 12 11:36 gmon.out
$ gprof ./main1
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
...
```

Esercizio

Provate adesso a fare il profiling del vostro programma di moltiplicazione matrice matrice.

Individuare e risolvere, se presente, il problema o i problemi nei programmi testdb e testdb1 dopo aver creato makefile opportuni alla compilazione.