Getting rid of LibC

Utilizziamo le chiamate di sistema al posto della LibC

Requisiti: conoscenza linguaggi C e assembly per architettura intel x86, architettura sistemi unix.
Obiettivo: scrittura di un programma in C senza dipendere dalla LibC né durante la compilazione né durante l'esecuzione.
Piattaforma software utilizzata: Freebsd 7.2 per i368, gcc, nasm, make, ld, ldd

I kernel unix mettono a disposizione dello spazio utente delle funzioni (*chiamate di sistema*, o in inglese *system calls*, abbreviato spesso in ***syscalls***), che hanno delle particolarità molto interessanti.

1. Sono standard. Questo permette allo sviluppatore di usare queste funzioni come se fossero una vera e propria LibC portabile già insita nel sistema.
2. Usandole, chiamiamo direttamente il kernel. Perchè chiamare *printf* che va a sua volta a fare delle operazioni e finire col chiamare la *write*, quando possiamo chiamare direttamente la *write*?
3. Sono sicure. Nel senso che sono molto più controllate e revisionate di altro codice spazio utente. Questo perchè sono la porta di accesso al kernel, e la selezione naturale dei kernel le ha portate ad uno stato di sicurezza estrema. Naturalmente l'uso sbagliato di queste chiamate potrebbe aprire delle falle pericolose nel vostro sistema.
4. *Last but not the least,* sono accessibili da interrupt. Questo significa che possiamo richiamarle tramite codice Assembly con la famosa istruzione *INT $0x80*. Chi ha programmato in Assembly per DOS si ricorderà l'altrettanto famoso *INT 21h*.

Alla luce di tutto ciò possiamo finalmente parlare dello scopo di questo articolo: **imparare a usare le syscalls come se fossero la nostra LibC. Ma più veloce, più sicura, e *built-in***!

Poniamo di voler fare un semplice programma che chieda all'utente di inserire un numero e restituirci il suo quadrato:

quadrato.c:

```
/* quadrato.c */
#include <stdio.h>

int main(int argc, char*argv[]) {
        unsigned int a;
        printf("Numero: ");
        scanf("%d",&a);
        printf("%d\n",a*a);
        return 0;
}
```

```
%cc -Wall -o quadrato quadrato.c
%ls -la quadrato
-rwxr-xr-x  1 vlad  vlad  4841 Nov 16 00:25 quadrato
%ldd quadrato
quadrato:
        libc.so.7 => /lib/libc.so.7 (0x2807d000)
```

La dipendenza dalla LibC è evidente. Proviamo innanzitutto a compilare quadrato con il flag *-static*. Questo includerà staticamente la LibC (o parte di essa) nell'eseguibile, che aumenterà di dimensioni ma non dipenderà dalla LibC installata nel sistema in cui verrà eseguito:

```
%cc -Wall -o quadrato quadrato.c -static
%ls -la quadrato
-rwxr-xr-x  1 vlad  vlad  226752 Nov 16 00:36 quadrato
%ldd quadrato
ldd: quadrato: not a dynamic ELF executable
```

Vediamo infatti che *ldd* ci dice che *quadrato* non è un eseguibile con dipendenze dinamiche. Questo però non è un modo intelligente di risolvere i nostri problemi: non abbiamo bisogno della LibC installata nel sistema, ma ne abbiamo comunque bisogno in fase di compilazione.

Consideriamo che questa libreria ci serve per due sole funzioni: *printf* e *scanf*. Queste sono dei casi particolari di *fprintf* e *fscanf* quando hanno come primo argomento rispettivamente il file *standard output* e il file *standard input*. Useremo perciò al loro posto le due chiamate di sistema *write* e *read*.

quadrato2.c:

```
// quadrato2.c
int main() {

        int pos = 0;
        int resto;
        unsigned int a;
        char buffer[512];
        char strNumero[] = { 'N', 'u', 'm', 'e', 'r', 'o', ':', ' ', '\0' };
        write(0,strNumero,9);
        read(1,buffer,512);

       //trasformo i caratteri ricevuti in numero
        a=0;
        while(*(buffer+pos)>='0' && *(buffer+pos)<='9') {
                a*=10;
                a+=(unsigned int)(*(buffer+pos)-'0');
                pos++;
        }

        //elevo il numero al quadrato
        a=a*a;
        //ritrasformo il numero in caratteri ascii
        pos=0;
        do {
                resto = a%10;
                buffer[pos]=(char)resto + '0';
                pos++;
                a = a-resto;
                a /= 10;
        } while (a!=0);

        char temp;
        int i=0;
        for (i=0; i<=(pos-1)/2; i++) {
                temp=buffer[i];
                buffer[i]=buffer[pos-i-1];
                buffer[pos-i-1]=temp;
        }
        buffer[pos]='\n';
        // lo stampo
        write(0,buffer,pos+1);

        return 0;
}
```

Getting rid of LibC

Utilizziamo le chiamate di sistema al posto della LibC

```
%cc -Wall -o quadrato2 quadrato2.c -static
quadrato2.c: In function 'main':
quadrato2.c:10: warning: implicit declaration of function 'write'
quadrato2.c:11: warning: implicit declaration of function 'read'
```

Come vediamo, abbiamo usato con vari espedienti le chiamate *write* e *read*, tuttavia ancora abbiamo bisogno della libc, anche se non la usiamo esplicitamente, per delle "funzioni strane" come *_init_tls*, ecc. il cui significato va oltre gli scopi di questo articolo. Se proviamo a non includere la LibC, infatti,

```
%cc -Wall -o quadrato2 quadrato2.c -static -nodefaultlibs
quadrato2.c: In function 'main':
quadrato2.c:10: warning: implicit declaration of function 'write'
quadrato2.c:11: warning: implicit declaration of function 'read'
/usr/lib/crt1.o(.text+0x61): In function `_start':
: undefined reference to `atexit'
/usr/lib/crt1.o(.text+0x6d): In function `_start':
: undefined reference to `atexit'
/usr/lib/crt1.o(.text+0x8d): In function `_start':
: undefined reference to `exit'
/usr/lib/crt1.o(.text+0x92): In function `_start':
: undefined reference to `_init_tls'
/var/tmp//ccnKNAWB.o(.text+0x74): In function `main':
: undefined reference to `write'
/var/tmp//ccnKNAWB.o(.text+0x92): In function `main':
: undefined reference to `read'
/var/tmp//ccnKNAWB.o(.text+0x21c): In function `main':
: undefined reference to `write'
```

possiamo notare ciò di cui abbiamo ancora bisogno. Per poter usare *-nodefaultlibs* dovremmo implementare ognuna di queste funzioni.

Cosa sta succedendo?

Come accennato in precedenza, la LibC non contiene solo quelle famose funzioni standard (*printf*, *scanf*, ...) che ci piacciono tanto, ma anche

1. I wrapper per tutte le chiamate di sistema
2. Le "funzioni strane" come *atexit* e *_init_tls* che devono occupare dei posti specifici nell'header del file eseguibile.

Ci sembra di essere in un vicolo cieco. ***Tuttavia, se riscrivessimo il programma tutto in assembler, avremmo la prova che è possibile fare quello che vogliamo senza dipendere dalla LibC.*** Sembra allora che l'infrastruttura di gcc sia "troppo legata" alla presenza della LibC.

La soluzione ai nostri problemi è semplice: creare un programma assembly che

1. Definisca i wrapper per le nostre chiamate di sistema.
2. Costituisca l'*entry point* del programma e richiami la nostra funzione *main* del programma in c: in questo modo eliminiamo i riferimenti a *__init_tls* e altre "funzioni strane", perdendo però vantaggi strutturali come ad esempio la possibilità dell'uso di *atexit*.

Possiamo creare una struttura di file, quindi, di questo tipo:

1. Aiuto nella compilazione:
   - Makefile
2. Parte assembly:
   - system.inc
   - syscall_wrapper.inc
   - syscall_wrapper.asm
3. Parte C:
   - syscall_wrapper.h
   - quadrato2.c

Andiamo a vedere i due file che includiamo nel sorgente assembly.

system.inc:

```
;
; system.inc
;
; contiene macro e
; definizioni
; che ci agevolano nella
; scrittura del programma
;


; definiamo i file di sistema
%define     stdin 0
%define     stdout      1
%define     stderr      2

; definiamo le syscalls
%define     SYS_nosys   0
%define     SYS_exit    1
%define     SYS_fork    2
%define     SYS_read    3
%define     SYS_write   4
[...]


; definiamo il modo
; per chiamare il kernel

%macro      system      1
      mov   eax, %1
      call
access.the.bsd.kernel
%endmacro


; definiamo le macro
; per chiamare le syscall
%macro      sys.exit    0
      system      SYS_exit
%endmacro

%macro      sys.fork    0
      system      SYS_fork
%endmacro

[...]

; chiamata al kernel
section     .text
align 4
access.the.bsd.kernel:
      int   80h
      ret
```

Getting rid of LibC

Utilizziamo le chiamate di sistema al posto della LibC

syscall_wrapper.inc:

```
;
; syscall_wrapper.inc
;
; contiene i wrapper
; per ogni chiamata
; di sistema
;

section .text

align 4
write:
        push    ebp
        mov     ebp,esp

        push    dword [ebp+16]
        push    dword [ebp+12]
        push    dword [ebp+8]
        sys.write

        mov     esp,ebp
        pop     ebp
        ret

align 4
read:
        push    ebp
        mov     ebp,esp

        push    dword [ebp+16]
        push    dword [ebp+12]
        push    dword [ebp+8]
        sys.read

        mov     esp,ebp
        pop     ebp
        ret

align 4
exit:
        push    ebp
        mov     ebp,esp

        push    dword [ebp+8]
        sys.exit

        mov     esp,ebp
        pop     ebp
        ret

[...]
```

Adesso possiamo vedere il codice di *syscall_wrapper.asm*, che contiene la vera *entry point* del programma:

syscall_wrapper.asm:

```
;
; syscall_wrapper.asm
;
; contiene l'entry point
; dell'eseguibile generato
;

; includiamo alcune definizioni
; e alcune macro che ci agevolano
; nella scrittura del programma
%include        'system.inc'

; diciamo al linker che esiste una
; funzione denominata 'main'
; all'esterno di questo file
extern main

; sezione dati
section .data

; sezione codice
section .text
; Dichiariamo 'globali' le funzioni
; accessibili da altri file e la
; _start.
; Queste funzioni corrisponderanno
; alle nostre chiamate di sistema
        global  _start
        global  exit
        global  write
        global  read

; includiamo i wrapper per le
; chiamate di sistema
%include 'syscall_wrapper.inc'

; entry point del programma in
; fase di esecuzione
_start:

        ; chiamiamo subito
        ; la funzione main del
        ; file c
        call main

        ; in eax abbiamo il valore
        ; di ritorno della funzione
        ; main e lo possiamo usare
        ; come valore di ritorno
        ; del programma
        push    dword eax
        sys.exit
```

Adesso andiamo a vedere di cosa abbiamo bisogno per *gcc*. Innanzitutto, vediamo *syscall_wrapper.h*:

syscall_wrapper.h:

```
/*
        syscall_wrapper.h
        contiene le definizioni
              'extern'
        alle funzioni wrapper
        delle chiamate di sistema
*/

#ifndef SYSCALL_WRAPPER
#define SYSCALL_WRAPPER

#include <unistd.h>

extern void
                exit(int rval);

extern ssize_t
                read(int fd, const
void *buf, size_t nbyte);

extern ssize_t
                write(int fd, const
void *buf, size_t nbyte);
[...]

#endif
```

e finalmente il nostro programma, *quadrato2.c*, identico a quello visto prima, tranne che per l'inclusione del file precedente:

quadrato2.c:

```
// quadrato2.c
#include "syscall_wrapper.h"
int main() {
[...]
```

Per capire come funzionerà questo framework, guardiamo il *Makefile*:

Makefile:

```
# Makefile
# modificalo a tuo piacimento!

CC              = gcc
CFLAGS          = -c -Wall -nostart-
files -nodefaultlibs
LDFLAGS         = -s
PROGRAM_NAME    = quadrato2
SOURCES         = quadrato2.c
OBJECTS         = $(SOURCES:.c=.o)

all:    $(SOURCES) $(PROGRAM_NAME)

$(PROGRAM_NAME): $(OBJECTS)
        nasm -f elf \
        syscall_wrapper.asm \
        ld $(LDFLAGS) -o
$(PROGRAM_NAME) syscall_wrapper.o\
$(OBJECTS)

.c.o:
        $(CC) $(CFLAGS) $< -o $@

clean:
        find . -name \*.o -exec rm
{} \;
        find . -name $(PROGRAM_NAME)
-exec rm {} \;
```

Come vediamo si compila il sorgente in C creando un file oggetto ".o", si compila il *syscall_wrapper.asm* creando il file oggetto *syscall_wrapper.o* e si crea il programma linkando questi due file insieme.
Lanciando *make* possiamo vedere come verrà creato un eseguibile funzionante, indipendente dalla LibC in fase di compilazione ed esecuzione.
E' importante non fare lo sbaglio di "portarsi avanti col lavoro" scrivendo i wrapper per tutte le chiamate di sistema, anche quelle che non usiamo nel programma in c. In questo modo vanificheremmo lo scopo di questo lavoro, ovvero "*avere nell'eseguibile solo quello che ci serve*". ▮

Giovanni Vergine
verginegiovanni@gmail.com

Fonti: http://www.int80h.org (per l'uso dell'assembly sotto sistemi FreeBSD), pagine man dei programmi utilizzati.